MAKE | BUILD | HACK | CREATE

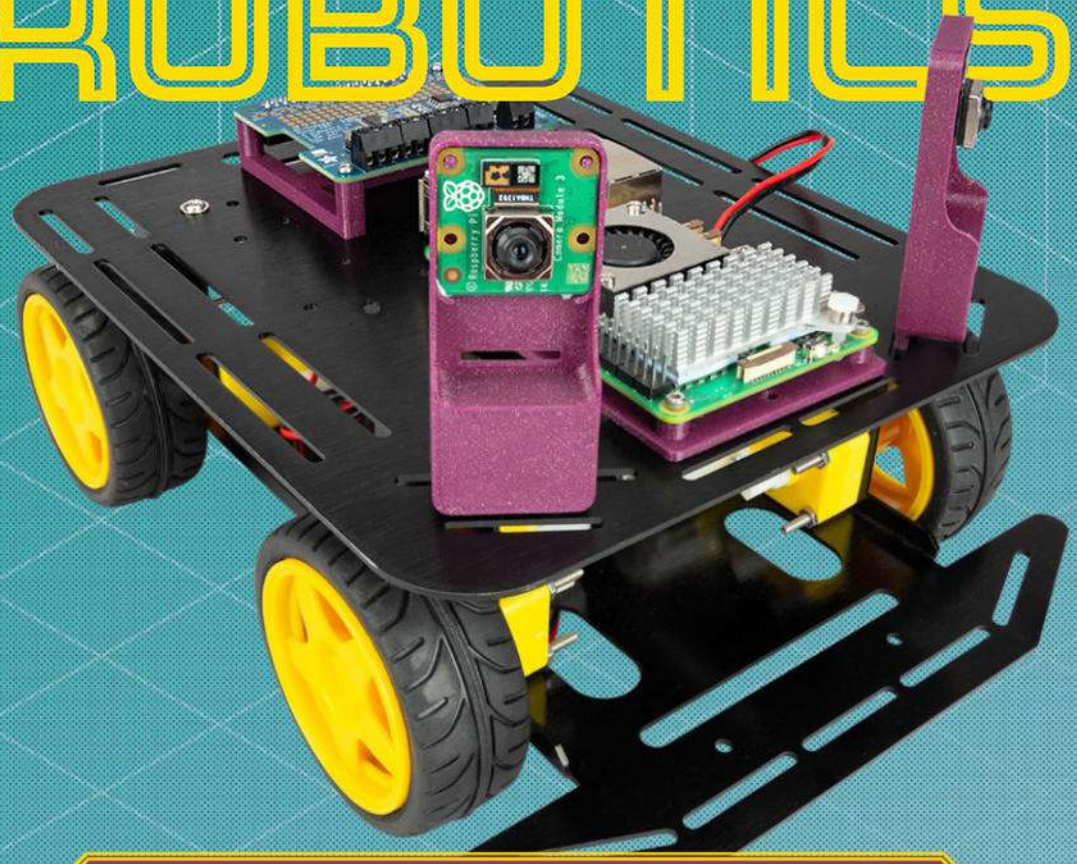
HackSpace

TECHNOLOGY IN YOUR HANDS

hsmag.cc | December 2023 | Issue #73

Raspberry Pi 5

ROBOTICS



Build and program an AI rover

JUKEPHONE
Upgrade old tech with a Raspberry Pi Pico

RETRO ROTARY
Turn an ancient phone into a personal assistant

MATT VENN
Zero to ASIC: you too can design a computer chip

Dec. 2023
Issue #73 £6

9 772515 514006 73

ARCADE GAMES **RESIN** DIY SYNTH

HiPi.io

PiKVM

Manage your servers or workstations remotely

A **cost-effective** solution for data-centers, IT departments or remote machines!

PiKVM HAT
for DIY and custom projects

Pre-Assembled version

- Real-time clock with rechargeable super capacitor
- OLED Display
- Bootable virtual CD-ROM & flash drive
- Serial console
- Open-source API & integration
- Open-source software

Available at the main Raspberry Pi resellers

core ELECTRONICS · PiShop.ca · RPishop.cz · Welectron. · elektorstore · PIMORONI

Pi-Shop.ch · ThePiHut · PiShop.us · CloudFree · PiShop.mx

Reseller suggestions and inquiries:
wholesale@hipi.io

Welcome to HackSpace magazine

Robots are strange things because, in many ways, they've completely taken over our lives. But, on the other hand, it seems like they haven't because as soon as a robot becomes common, we stop calling it a robot. 3D printers, automatic vacuum cleaners, even many cars, are all robots. They're machines that are controlled directly by a computer. However, we don't usually call them robots. This issue, we're not skirting around the fact. We're making robots, and we're calling them exactly that.

Raspberry Pi 5 makes a great base for a robot because it's got enough processing power to churn through almost any sensor input you can throw at it. In this issue, we're going to test this out by processing image data coming in from two cameras.

If that's not enough, we're also going to work on the other end of the spectrum and build a robot with Pico.

We'll get you started with the basics, and you can customise them into whatever role you like. Who knows, your project might be successful enough that people stop calling it a robot.

BEN EVERARD
Editor ben.everard@raspberrypi.com

Got a comment, question, or thought about HackSpace magazine?

get in touch at **hsmag.cc/hello**

GET IN TOUCH

hackspace@raspberrypi.com
hackspacemag
hackspacemag

ONLINE

hsmag.cc

Available on the App Store

GET IT ON Google Play

PAGE 34

EDITORIAL

Editor
Ben Everard
ben.everard@raspberrypi.com

Features Editor
Andrew Gregory
andrew.gregory@raspberrypi.com

Sub-Editors
David Higgs, Nicola King

DESIGN

Critical Media and Raspberry Pi
criticalmedia.co.uk

Head of Design
Lee Allen

Designers
Sam Ribbits, Sara Parodi, Jack Willis

Photography
Brian O'Halloran

CONTRIBUTORS

Marc de Vinck, Jo Hinchliffe, Thomas Burns, Rob Miles, Nicola King, Turi Scandurra

PUBLISHING

Publishing Director
Brian Jepson
brian.jepson@raspberrypi.com

Advertising
Charlie Milligan
charlotte.milligan@raspberrypi.com

DISTRIBUTION

Seymour Distribution Ltd
2 East Poultry Ave,
London EC1A 9PT
+44 (0)207 429 4000

SUBSCRIPTIONS

Unit 6, The Enterprise Centre,
Kelvin Lane, Manor Royal,
Crawley, West Sussex, RH10 9PE

To subscribe
01293 312189
hsmag.cc/subscribe

Subscription queries
hackspace@subscriptionhelpline.co.uk

recycle

This magazine is printed on paper sourced from sustainable forests. The printer operates an environmental management system which has been assessed as conforming to ISO 14001.

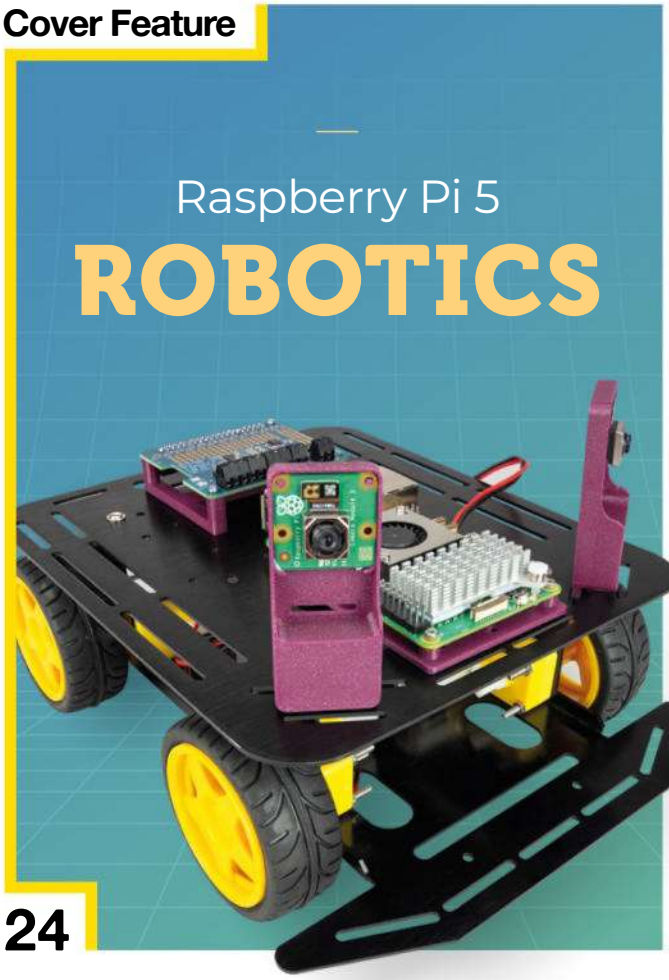
Contents

06

06 SPARK

23 LENS

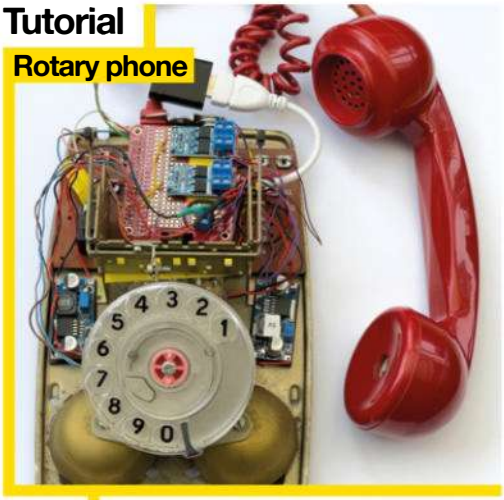
Cover Feature



24

Tutorial

Rotary phone



70 Build a home assistant with Neolithic technology – the humble rotary phone

36

Interview

Matt Venn

Crowdfunding

Open UpCell

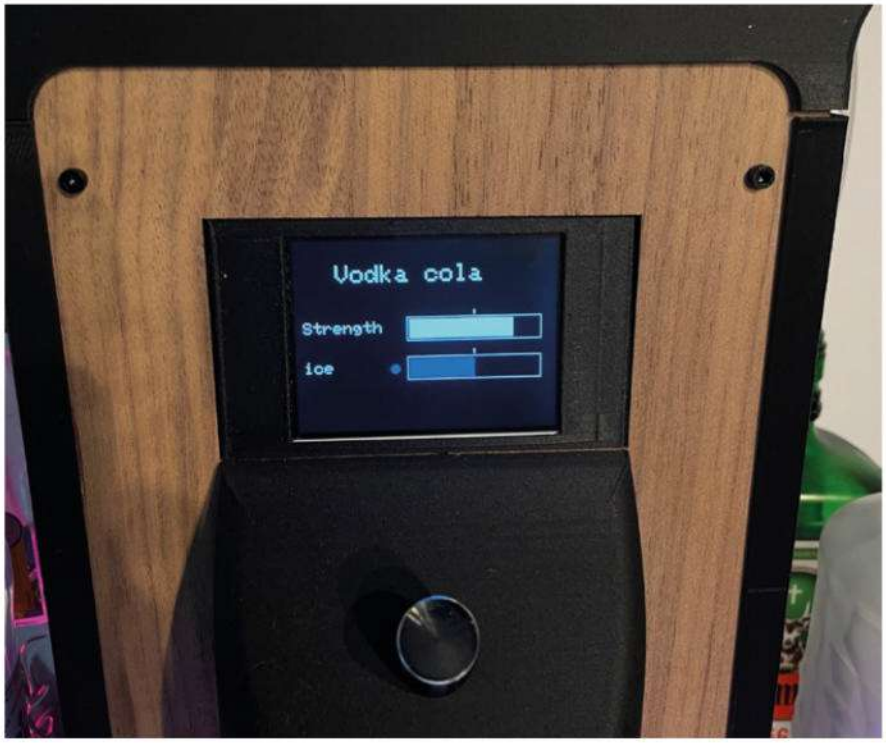
53 FORGE

85 FIELD TEST

Arduino cocktail machine

By **Sven Kroesen** hsmag.cc/CocktailMachine

Christmas is coming, which is a good excuse to gather socially and drink fermented vegetable products. And what better way than via this Arduino-powered cocktail dispenser, which features a rotary encoder for navigating the cocktail menu, a TFT screen, laser-cut plywood and plastic body, and an Archimedes screw to give your beverage just the right amount of ice. It's brilliant: so brilliant that we've asked Sven to tell us all about how he built it in the next issue of HackSpace.



Right
Why spend minutes mixing a cocktail when you can spend months automating it?


THE ORIGINAL
GORDON'S
LONDON DRY GIN
IMPORTED
AWARD WINNING
WHITE RUM
BACARDÍ ESTABLECIDO EN CUBA EN 1862
EL COCO
THE LEGENDARY PALM
SUPERIOR
WHITE RUM
37.5% alc./vol.

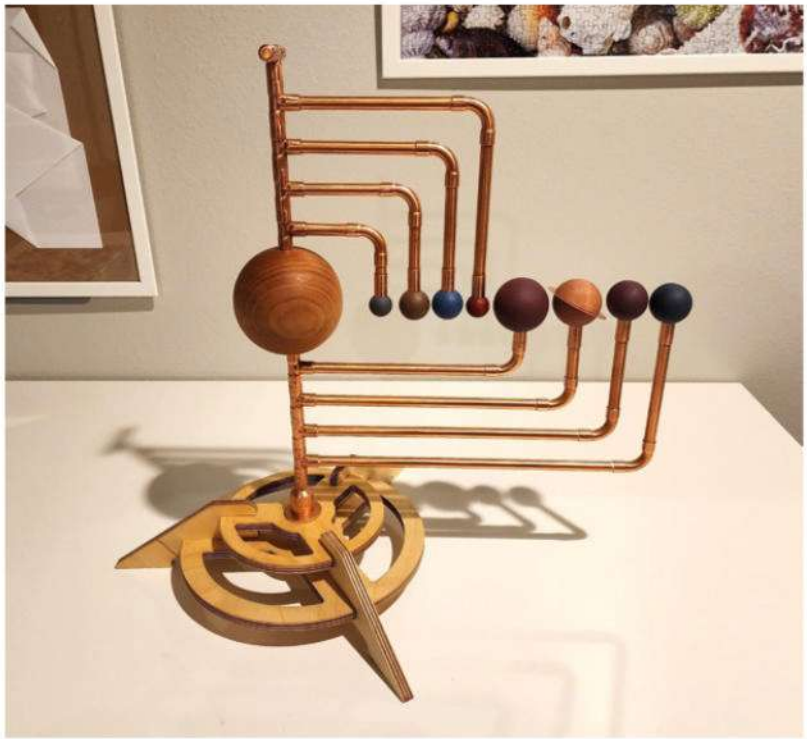
REGULAR

Orrery

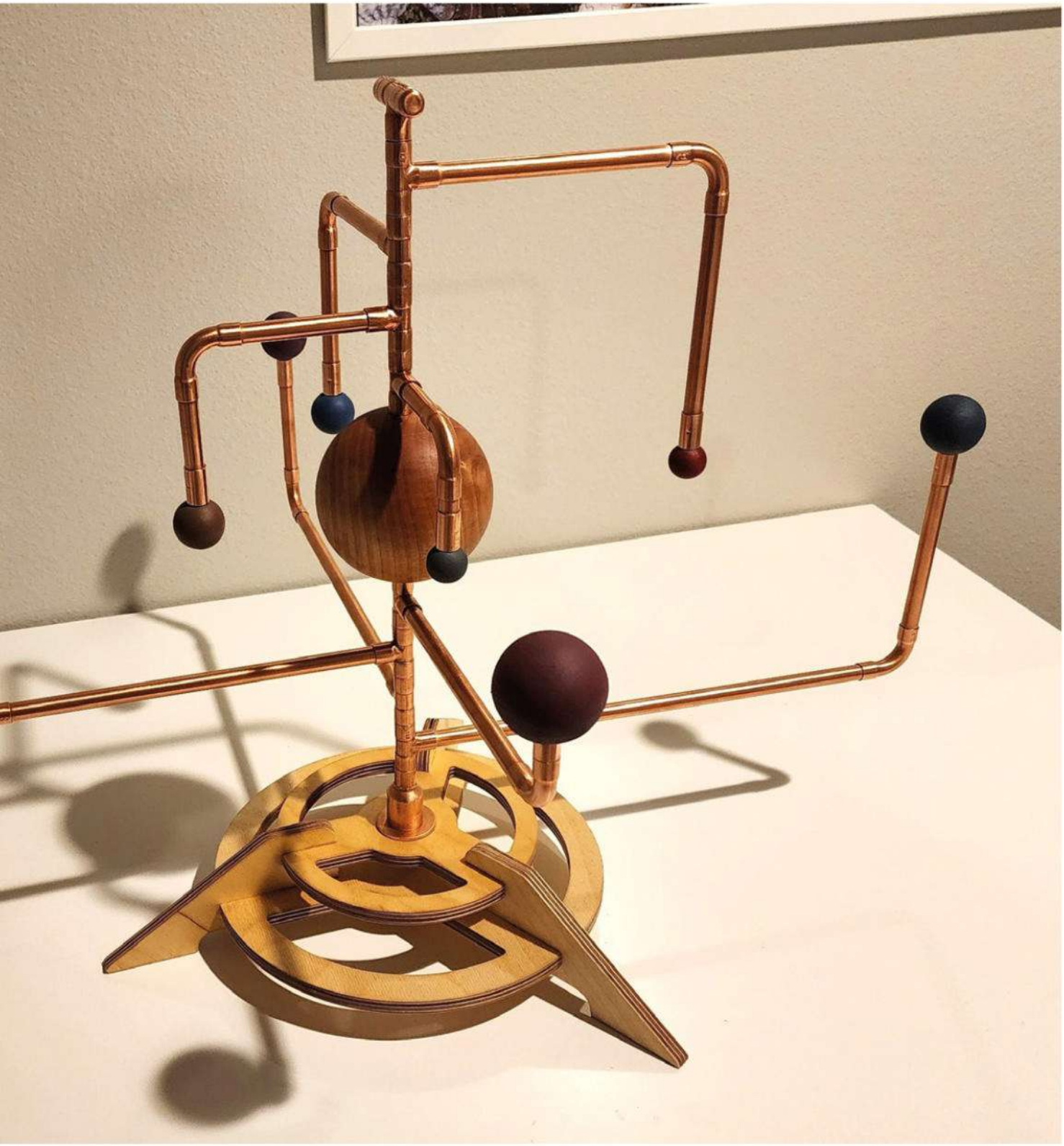
By MarkH342 hsmag.cc/PipeOrrery

Building an orrery is a brilliant way to combine the theoretical complications of maths and astrophysics with the practical elements of woodwork, laser cutting, or whatever other method of making you're most comfortable with. In this case, the maker has used copper pipes to support the planets on their way around the sun. And it works brilliantly.

There's no motor or computerised element to this build: it's just a laser-cut plywood base, wooden balls to represent the planets, and that glorious copper piping, T-joints, elbow joints, and end caps, all rotating around a central steel rod.



Right
Mark used no-heat solder to join the copper parts together. We had no idea such a thing existed!



REGULAR

TechNIK's Cyberdeck

By **Nik Reitmann** hsmag.cc/TechNIK-cyberdeck

We love seeing Raspberry Pis built into fresh new packages. Nik Reitmann's cyberdeck follows a solid, sturdy design that reminds us of the beige box that used to get wheeled into the classroom for our regular one hour of early 1990s computing.

It's based on a Raspberry Pi 4, and the design features a trackball rather than a trackpad, to save space; it can run DOOM; it can access the internet over Wi-Fi; and the creator has broken out eight of the Raspberry Pi 4's GPIO pins for easy breadboard tinkering.

This build really shines in its execution; all the screws used in construction are internal, giving it the clean lines of an injection-moulded product, and there's even an extra usability feature in the shape of a scroll-wheel connected to a rotary encoder, for quickly moving up and down text documents.

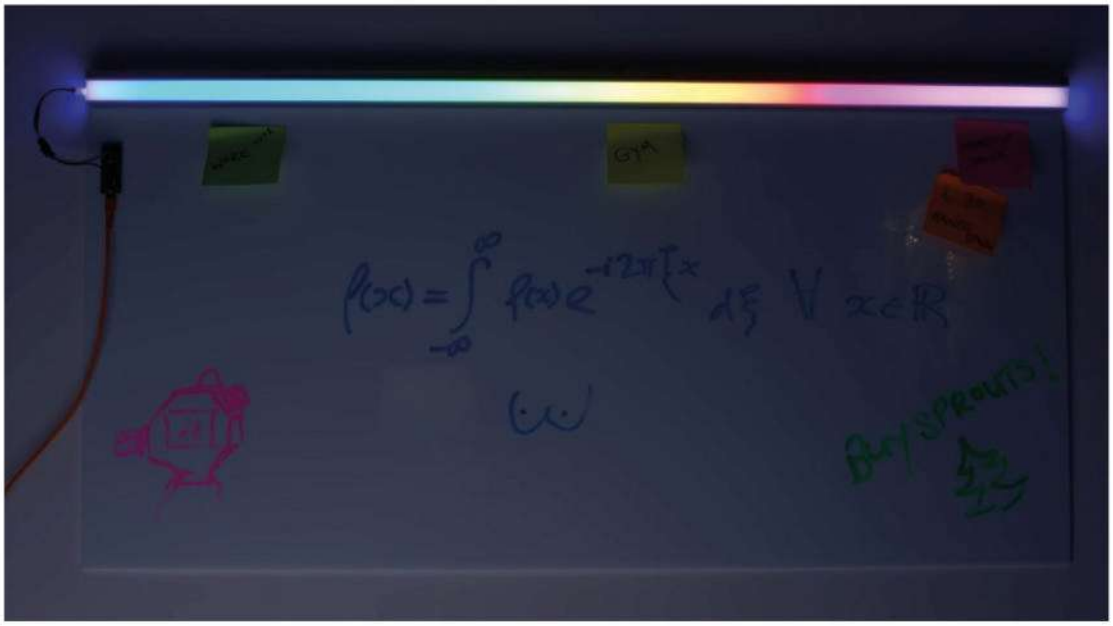
Right That gorgeous screen is a Waveshare 7.9inch HDMI LCD


Del
Enter
Shift
Fn1
Ctrl
Alt
Win
Fn

Calendar progress bar

By **VEEB Projects** hsmag.cc/ProgressBar

Here's one for the minimalists: a progress bar for your day. It's a simple, intuitive display that displays a line of light that grows from left to right as the working day inches by. And because it's connected to the internet via a Raspberry Pi Pico W and the magic of MicroPython, it can grab data from your Google Calendar and display when your day's events are. Other than the Pico W, the components are simple: just a strip of RGB LEDs, a frame to hold them (angled aluminium from your local DIY shop would work), plus a strip of opaque plastic to work as a diffuser.



Above
The progress bar can flash to alert the user to upcoming events

Nerdy Gurdy

By **Kudlas** hsmag.cc/NerdyGurdy

Now this is a thing of beauty. Instructables user Kudlas is a fan of medieval music – the troubadours, the chivalry, the Plantagenet romanticism of it all. He also knows his way around a laser cutter, which is where this creation was born.

While there are kits available, Kudlas chose to make this from scratch, starting with a template from the internet which he modified to produce a superior result. The body is a combination of 3 mm and 6 mm Baltic birch plywood, and there are standard guitar tuners, screws, and a few 3D-printed parts. Where Kudlas deviated from the usual build is that, instead of using a threaded rod to rotate the wheel, he used a smooth piece of linear rod, of the sort commonly seen in 3D printers, which he reckons should result in a smoother action when the player turns the wheel. He's also added a most excellent green sunburst paint job. Chaucer would approve.

Right Do you want a Nerdy Gurdy of your own? Head to nerdygurdy.nl to get started

A La QRTE

By **Guy Dupont** hsmag.cc/QRTE

Have you ever been to one of those restaurants that don't have a paper menu, instead inviting you to scan a QR code on your phone? If so, and if you found it as annoying as we do, you could take a leaf out of Guy Dupont's book. He's built this delightful machine, the A La QRTE, which scans the QR code, presses it into a simplified format, and prints out a menu on paper, as is right and proper.

The device runs on a 12 V battery, and uses a QR code reader from Useful Sensors, and an ESP-32 S3 from Seeed Studio to drive the printer module. There's also a little bit of Python involved that scrapes data from the web and formats it so that it's printable. We're on our phones enough as it is without using them when we just want a bit of something to eat, so we say this device is a wonderful slap in the face to an unwanted modern trend.

Right
This is a wonderfully passive-aggressive tool to make restaurant-going more awkward than it needs to be


Treats
Oatmeal Chocolate Chip Cookie $2
Banana Coffee Cake $2.85
Salted Butter +$1.25 / $6
Cream Cheese Spicy Pepper +$1.75 / $6.5
Mustard Butter +$1.25 / $6.5
Housemade Almond Butter +$2.5 / $8.5
Cream Cheese Honey Rosemary +$1.75/ $6.5

Objet 3d'art

3D-printed artwork to bring more beauty into your life

A chip – even a large, complicated one – is just a collection of switches. Ons and offs, AND gates and OR gates, inverters, adders: simple things that, when combined, enable hugely complicated flows of logic that control computers. You don't have to understand these in order to use a computer (or in order to use the machines that depend on computers), but if you're going to design a chip, it helps to know what these logical building blocks are. Shown here is a 3D-printed model of an inverter: a pair of transistors that takes a signal in (either a 1 or a 0) and converts it to its opposite (a 0 or a 1). Matt Venn showed it to us when we spoke to him about his Zero to ASIC course – thanks to him, we're now slightly less baffled about how the world works.

zerotoasiccourse.com

Letters

ATTENTION ALL MAKERS!

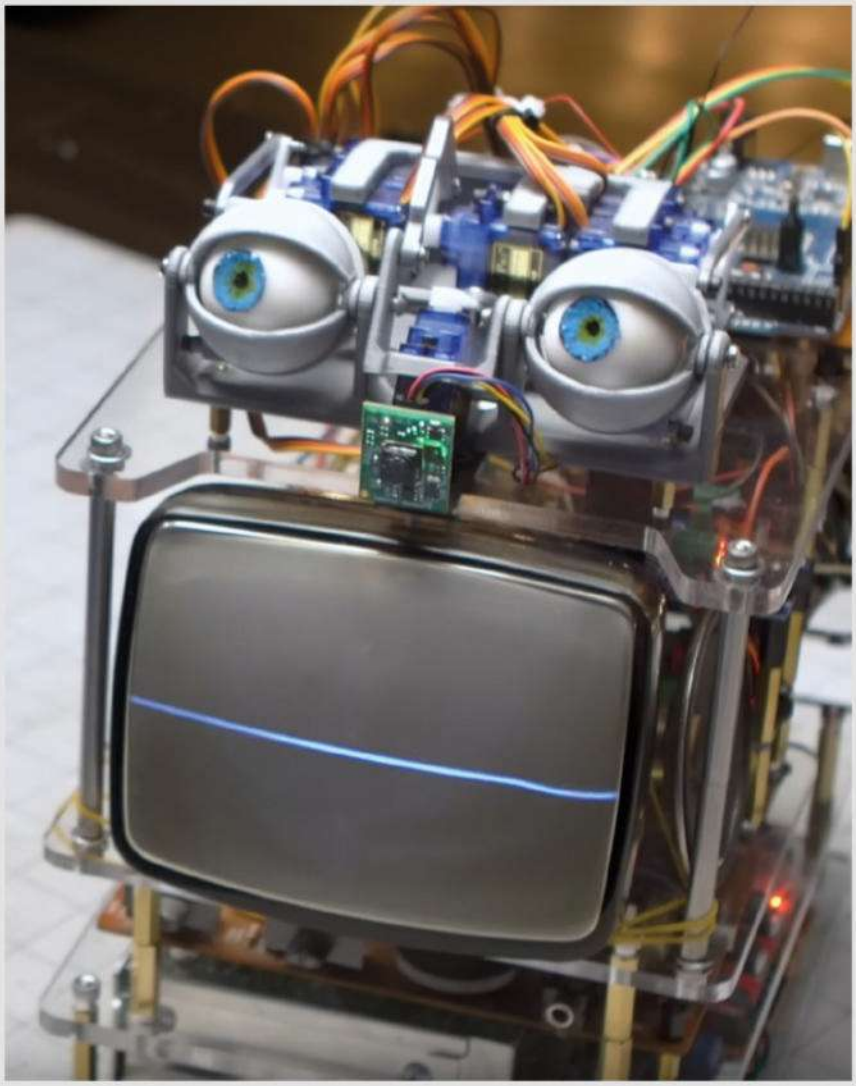
If you have something you'd like to get off your chest (or even throw a word of praise in our direction), let us know at **hsmag.cc/hello**

ALEXATRON

We're lucky to live in a time when things are easy for makers. Unfortunately, this ease can sort of pigeonhole us into certain things. There are some off-the-shelf screens that are really easy to use, which is great, but it means that everyone's projects end up looking the same because they've used the same screen.

It was lovely to see Thomas Burns turning away from this and using an old-fashioned cathode-ray tube. I'm not sure if he'll see this, but if you're reading Thomas, it brought a simile to my face, so thanks.

Aaron
Leeds



Ben Says: That really was a beautiful build. There is something a bit special about CRTs. I'm not sure if it's just looking back to my childhood through rose-tinted glasses, or if it's because everything modern is flat and square so the curved surfaces look slightly otherworldly, but seeing them still work brings a smile to my face too. Thanks, Thomas.

CARDBOARD FUTURE

I play with cardboard with my kids, and I've always thought it was a great material to use. However, when I read the article about invenTABLE, I instantly recognised the issue of cutting it. It IS a pain in the backside to cut with scissors. It's especially annoying when you have to fold it slightly and it adds creases where you don't want any.

I'm not sure that I'll stretch to the $189 price tag myself, but I hope that I'll be able to persuade my daughter's school, or perhaps the craft club at the local library, to purchase one.

Stephen
York

Ben Says: That's the beauty of shared spaces, be they hackspaces, maker spaces, libraries, or any other group. Tools that would be too expensive for an individual are much easier to acquire. For us grown-ups, that might be a table saw or laser cutter, but tools like this are great for young makers.

IN PRAISE OF JEFF

Can I just say that I'm a massive Jeff Geerling fanboy? There are plenty of excellent technical people in the world who can make computers do all sorts of fantastic things. There are also many excellent communicators in the world who can keep me reading or watching videos. However, there are very, very few people who fall into both camps. I have no real interest in hooking up a graphics card to a Raspberry Pi (I don't even use one in my desktop), however, I can quite happily watch Jeff try for hours on end, and I learn a bit about the intricacies of the Linux kernel along the way.

Joseph
Dover

Andrew Says: It's a rare gift to be able to understand something and also help others understand it, and we're lucky to have people like Jeff in our community who can do this. It's been a pleasure watching his videos over the years, and it was a pleasure speaking to him for the interview.

HiPi.io

PiKVM

Manage your servers or workstations remotely

A **cost-effective** solution for data-centers, IT departments or remote machines!

PiKVM HAT
for DIY and custom projects

Pre-Assembled version

- Real-time clock with rechargeable super capacitor
- OLED Display
- Bootable virtual CD-ROM & flash drive
- Serial console
- Open-source API & integration
- Open-source software

Available at the main Raspberry Pi resellers

core ELECTRONICS · PiShop.ca · RPishop.cz · Welectron. · elektorstore · PIMORONI

Pi-Shop.ch · ThePiHut · PiShop.us · CloudFree · PiShop.mx

Reseller suggestions and inquiries:
wholesale@hipi.io

HackSpace
TECHNOLOGY IN YOUR HANDS

LENS

HACK | MAKE | BUILD | CREATE

Uncover the technology that's powering the future

PG 36

HOW I MADE: JUKEPHONE

Play MP3s with big chunky buttons. Don't leave me hanging on the telephone!

PG 44

INTERVIEW: MATT VENN

You too can go from zero knowledge to building your own computer chip

PG 24

Raspberry Pi 5 ROBOTICS

Build your own intelligent robot with AI and the new Raspberry Pi

FEATURE

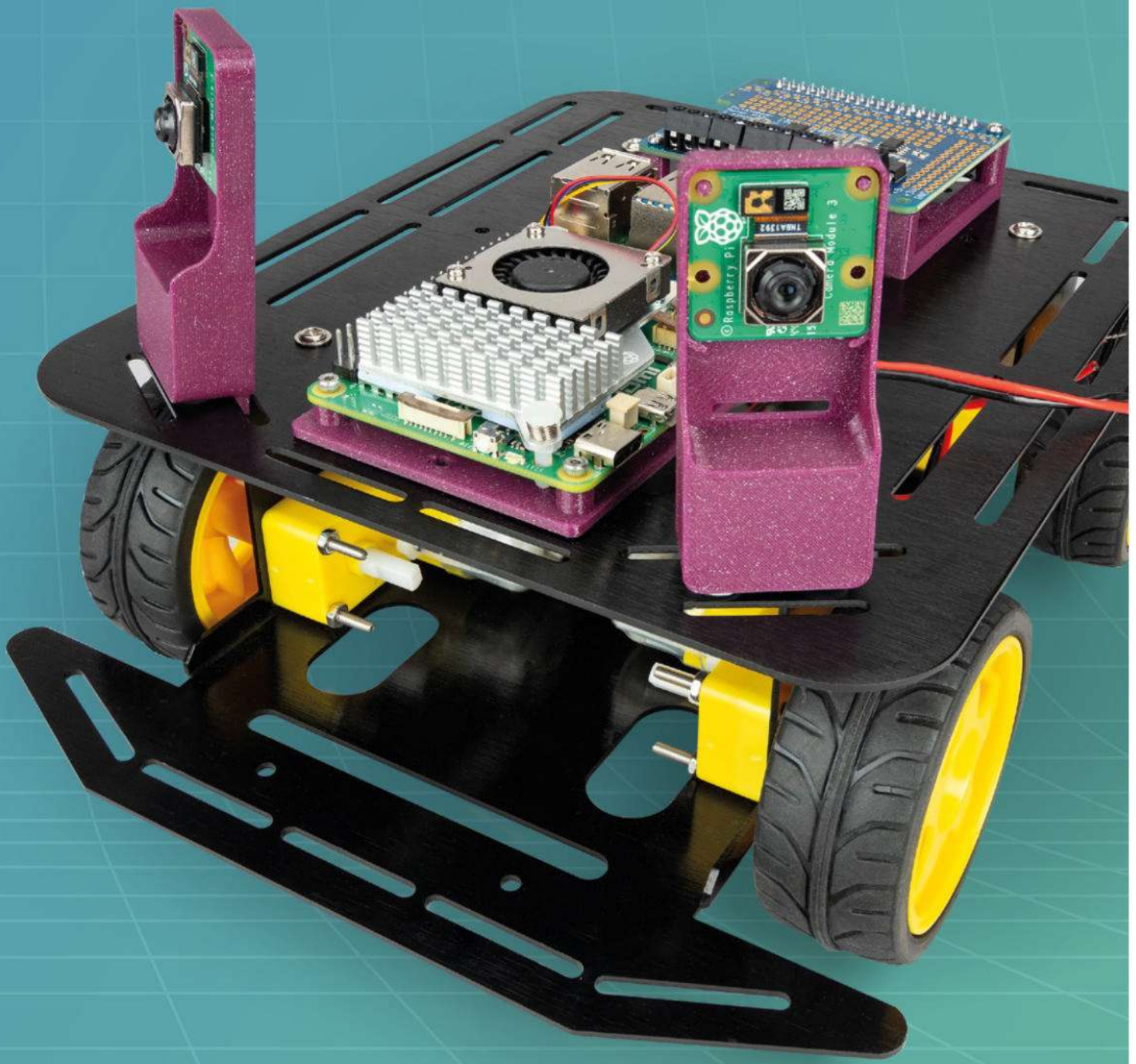
Raspberry Pi 5 ROBOTICS

Build and program an AI robot

The Raspberry Pi 5 is here and ready to supercharge your projects. We've looked at it in detail over the past two issues, but briefly, it's got more processing power, faster memory, and faster interfaces than previous models. Let's take a look at what this means to just one area: robotics.

You can drive motors with the puniest microcontroller. In fact, you can drive motors without any processing power at all and use sensors (such as light-dependant resistors) to directly control motor drivers and get interesting behaviour. However, with more computing power, you can do more complex things.

We're going to use two features of Raspberry Pi 5 to bring image recognition to our robot. Firstly, we're going to use both available camera slots to give our robot two eyes, and secondly, we're going to run both camera streams through a TensorFlow image recognition model. With Raspberry Pi 5's increased computation power, we can run both streams faster than we could run one previously.


Raspberry Pi
Camera Module 3

Pick your **hardware**

Giving your robot a body

Wheeled robots are simultaneously complex and simple. The basics of strapping a motor to a computational device and setting it running are usually fairly straightforward. Getting it to move in precisely the way you want, over the terrain you want, to the place you want, is a complex field of study that takes years to master.

Everything starts with the motors. These are what drive your robot, so everything else fits around them. What motors you want to use will determine which chassis you want and what electronics you need.

OTHER TYPES **OF MOTOR**

STEPPER MOTORS These motors turn a single step at a time (with a fixed number of steps per rotation). You can control them very precisely – for this reason, they're usually the type of motor controlling 3D printers and similar machines. They're a little more complex to use and more expensive than DC motors (though are becoming easier and cheaper).

CONTINUOUS ROTATION SERVOS A servo is a motor with a feedback mechanism to allow it to sense where it is. Usually servos can't turn a full rotation, and cover an arc of around 180 degrees. However, continuous rotation servos can turn all the way around. They can be easy to use, but are often slower than other types of motor.

AC MOTORS As the name suggests, these take alternating current rather than direct current, and are usually used for powerful mains-powered devices. As such, there are more risks with using them, and they're not commonly used in hobbyist robotics.

BRUSHLESS MOTORS These are basically the same as stepper motors but typically designed to be efficient. You can get very powerful small brushless motors, and they're used in everything from quadcopters to electric scooters. They need specific motor drivers (often called electronic speed controllers, or ESCs).

Small hobbyist robots usually use one of two categories of motor: plastic motors that are always (and for reasons we don't understand) yellow, and metal 'N20' motors. Both are 'DC' motors that will rotate continuously when a voltage is applied.

Plastic motors are cheap and ubiquitous. They come in a few different voltage options, so make sure that this matches what power supply you're planning to use. They also have a few different physical configurations. None of them are particularly small, but they fit differently onto the chassis. If you're planning on using an encoder, you'll need a version where the axle comes out of both sides of the motor housing.

N20 motors are a bit more expensive, smaller, and should be more robust. They also come in a range of voltage versions. N20 motors often come with an attached gearing, and the range can vary significantly. This can let you select whether you want a motor with a higher top speed or more torque.

As well as the type of motor, you need to decide on the number of them. Typically, this is two or four. With two motors, you can use a caster to balance the other end. With four, you can generate more power, and if you use mecanum wheels, get more complex movements. Four motors will also give you more control on carpet and other floors that aren't smooth.

No two motors are exactly alike, and this means that if you buy two of the same model, put them on either side of a robot, and power them in the

same way, the robot won't go straight. It'll do a gentle curve to one side or the other. You can minimise this in software, but you can't eliminate it. How big a problem this is depends on what you want your robot to do. If it's being directed by sensors, then it's often not a problem that it's not heading in exactly a straight line as it can continuously correct itself. However, if you do want it to go straight, the solution is encoders. These are sensors that attach to the wheel or motors that detect each time the wheel rotates. Using these, you can sense how much each wheel is moving and dynamically adjust the power. This isn't a perfect solution because they detect how much the wheel is rotating, but if the wheel loses traction, this won't necessarily correlate to how much the robot is moving.

Now you've got your motors, you'll need something to mount them on. You can buy a chassis off-the-shelf, or you can make one. The most complex bit is usually attaching the motors.

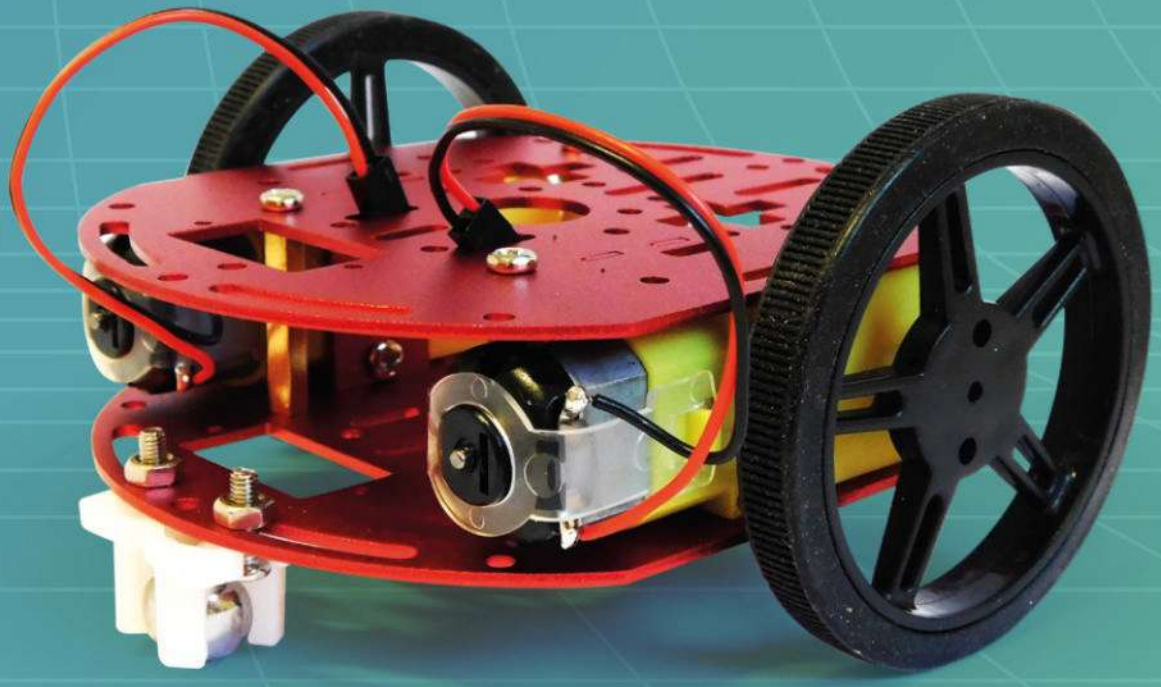
> **You can get purpose-made motor mounts, though this** can also be done with glue, tape, or cable ties

You can get purpose-made motor mounts, though this can also be done with glue, tape, or cable ties, if you're that sort of maker.

Obviously, your chassis has to provide enough space to mount the hardware you want. In our case, this is a Raspberry Pi 5, a motor driver board, two cameras, and two batteries.

We've opted to use yellow plastic motors on an off-the-shelf chassis that works with four motors. Two of the motors did come with encoders, though we're not using them for this project. This gives us a large, sturdy base on which to mount all our hardware.

Below
This chassis has two wheels and a caster. This configuration is easier and cheaper but a bit less stable and less powerful



FEATURE

Electronics

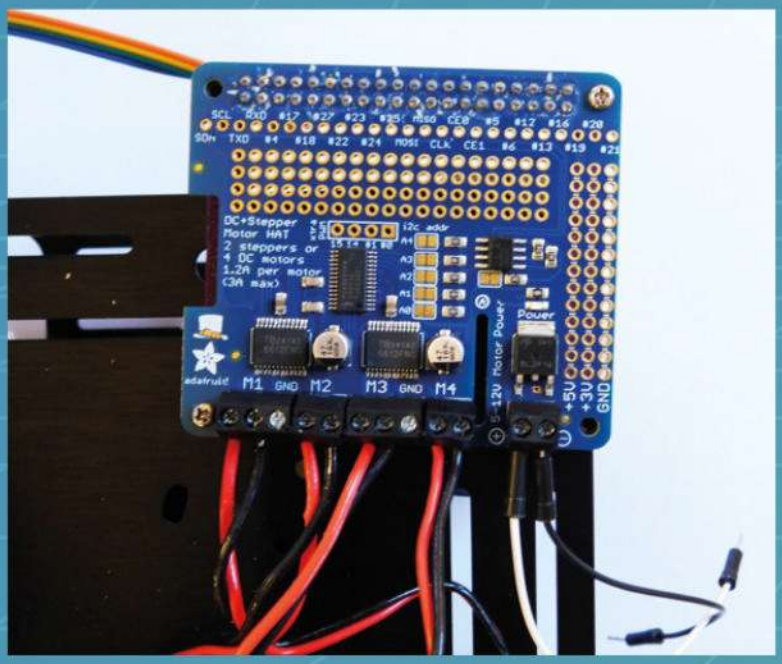
Control your bot

We're basing our robot on Raspberry Pi 5. You could use a different model of Raspberry Pi, though you wouldn't be able to have two cameras, and if it's a pre-version 4 Raspberry Pi, you might struggle to get the software to run at a sensible speed.

Alongside this controller, we're going to need a motor driver and power supplies.

Although Raspberry Pi 5 has a range of GPIO pins, these can only drive small currents, so can't power the motors directly. Instead, it has to go via a motor driver which can power larger currents and voltages. There are a lot of different motor driver boards out there. The three things that you want to consider when picking a motor driver board are:

Below
The four motors connect into the screw terminals on the HAT

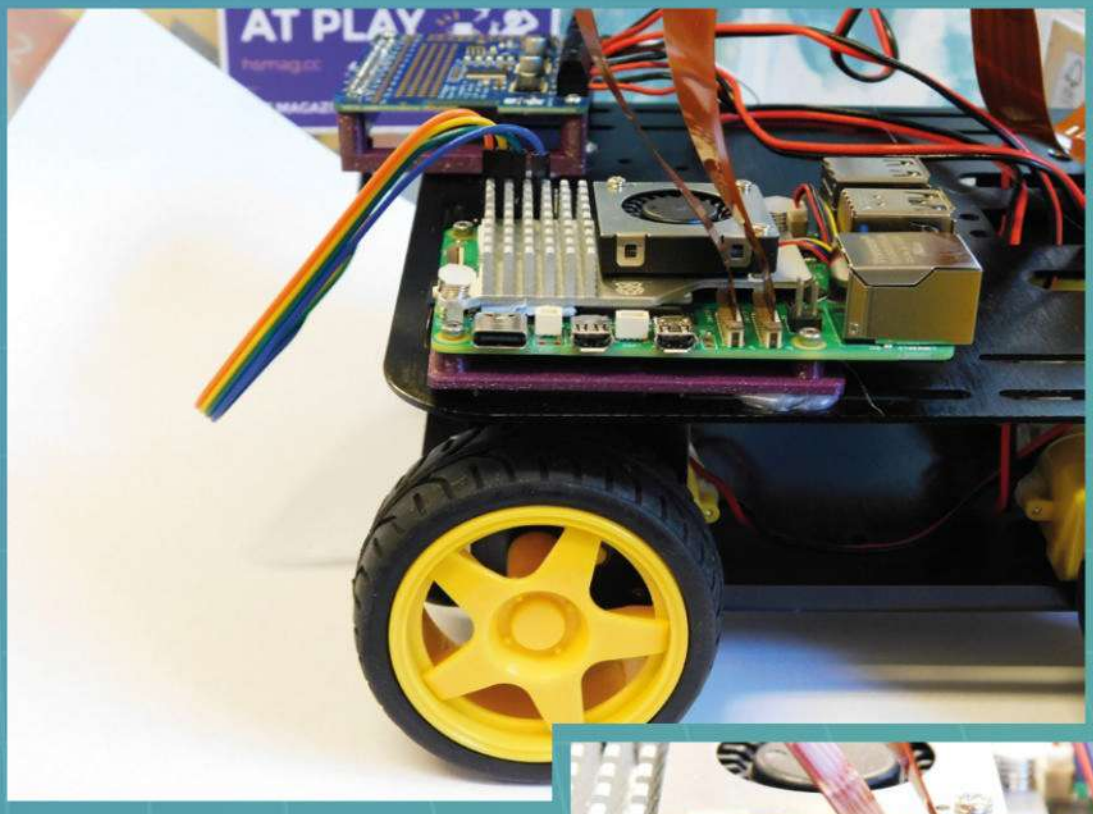


POWER Does it match the power of your motors? It has to be able to run at the voltage of your motors and handle (at least) the amount of current they consume. Current is given in amps, and it might be per channel (i.e. per motor) or across all motors. The more work motors have to do, the more current they will consume – with the maximum being the 'stall current', which is the amount of current they take if the motor is jammed and not moving. If you're unsure how much current your motors consume, you can hook them up to a power supply and measure it using a multimeter. The more force you apply to the motors as they spin, the more current they'll consume.

Most of the time, a motor driver will use an external power source (i.e. not the Raspberry Pi's power output). You need to match the voltage it can take here with both your battery's output and the voltage that the motors can take.

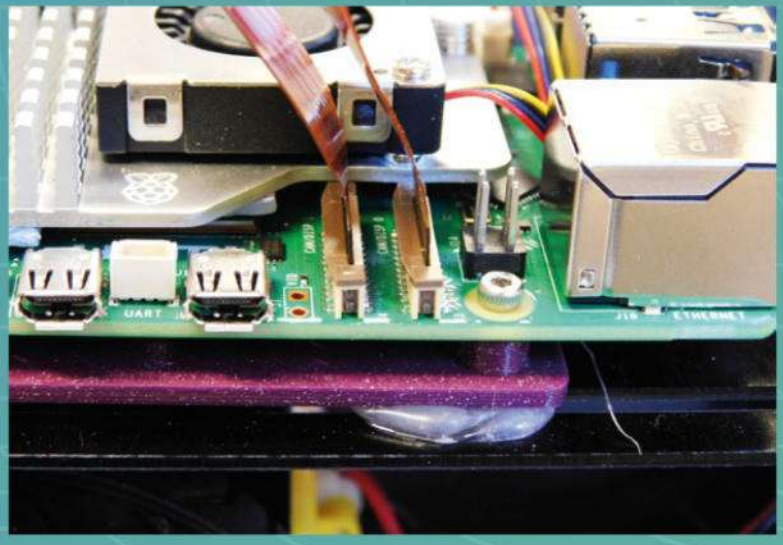
SOFTWARE SUPPORT Does it come with a way of controlling it using the programming language you'd like to use? We're going to look at Python, but you can use whatever you want. Some hardware manufacturers provide Python libraries, and some leave you on your own.

FULL H-BRIDGE You can control a motor by applying a voltage to it, and this can be done with something like a MOSFET or a relay, but a full H-bridge will give you the ability to go in reverse (and brake and coast, but the effectiveness of these depends on the motor). You will also want the ability to control speed, which is typically done using pulse-width modulation (PWM), so make sure your controller can support this.



Left
Hot glue is great for testing because it's quick, firm, but also does come apart

Below
It can be a bit fiddly to get the camera cables into the connectors. Take it slowly because they are delicate



NUMBER OF MOTORS This, fairly obviously, should be at a minimum the number of motors you plan to use. In some cases, it's possible to add more than one of the motor drivers to a Raspberry Pi, in some cases it's not, so if you plan on using multiples, check if the hardware and software support this.

> **You will also want the ability to control speed, which is** typically done using pulse-width modulation

ADDITIONAL HARDWARE You may well want to attach more hardware to your Raspberry Pi than just the motors and cameras. Does your motor controller support this? Some have additional hardware built in, some break out unused GPIO pins, and some include a prototyping area for you to add your own hardware to.

We opted for an Adafruit DC and Stepper Motor HAT. It's a good size, and can handle enough power for four motors. It's got an easy-to-use Python library, and although we didn't choose it because it had an available schematic, this did help us fix a problem while we were getting the robot wired up.

As well as a motor driver, you'll need power. It is possible to power both motors and Raspberry Pi from the same battery, but this can cause problems as the motors can create a lot of electronic noise on the power lines. It's a solvable problem, but we chose to sidestep it entirely by using two power supplies – a rechargeable USB-C battery for the board, and six AA batteries for the motors.

FEATURE

Putting it **all together**

Adding the intelligence

We've got our robot hardware and electronics. Let's put it all together and get our robot running. One slight issue with Raspberry Pi 5 is that the layout is a bit different to previous versions. Firstly, we want to make sure there's space for the active cooler; secondly, the ribbon connectors for the cameras are in slightly different positions. This means we can't just plug the HAT in normally.

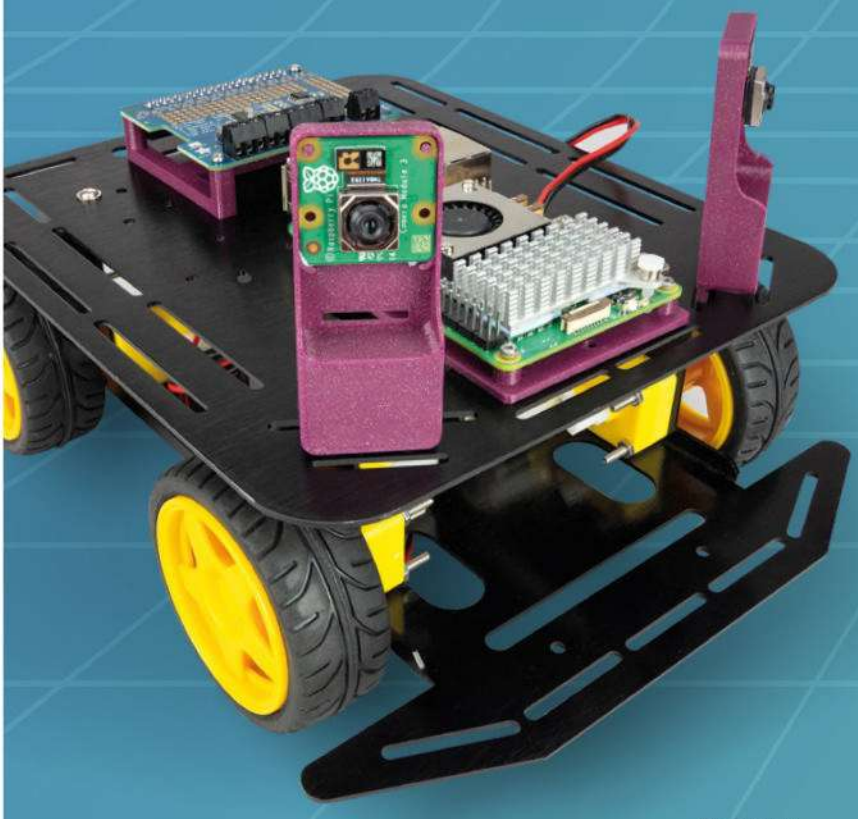
We could raise up the HAT, but it would have to be pretty high to allow the camera cables to pass underneath. Instead, we opted to simply use header wires to join the GPIO pins and the HAT. It's not always obvious which GPIO pins are used by a HAT, but fortunately, Adafruit releases the schematics, so we can take a look and see that it's 3.3V, GND, GPIO 2, and GPIO 3. Join those together and everything works as expected.

The chassis we have has a lot of mounting holes, but none quite right for our Raspberry Pi and HAT, so we 3D-printed a couple of mounts. We could have added screw holes for these, but a few drops of superglue served the job just as well.

The only part left is the cameras. Obviously the big question is where to point them. One option is to have them both pointing forwards to create a stereoscopic image that can be used for depth perception. We have experimented with this, and will look at it more in the future. For this robot, we opted to place the cameras facing diagonally out to give the robot a huge field of vision. This is similar to how many prey animals have their eyes. Look at a sheep, horse, or cow and their eyes face sideways so that they can take in a much greater range of the surrounding landscape.

To mount the cameras, we 3D-printed a camera mount (**hsmag.cc/cammount**) which, by default, has a tripod mount, but we added on a 3mm thick rectangle to slot into the rails on the chassis.

That's the hardware all set up, let's now take a look at the software.

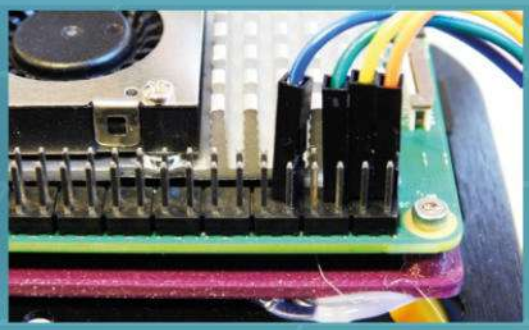


Below
The outward diagonal placement of the cameras gives the robot a wide range of vision

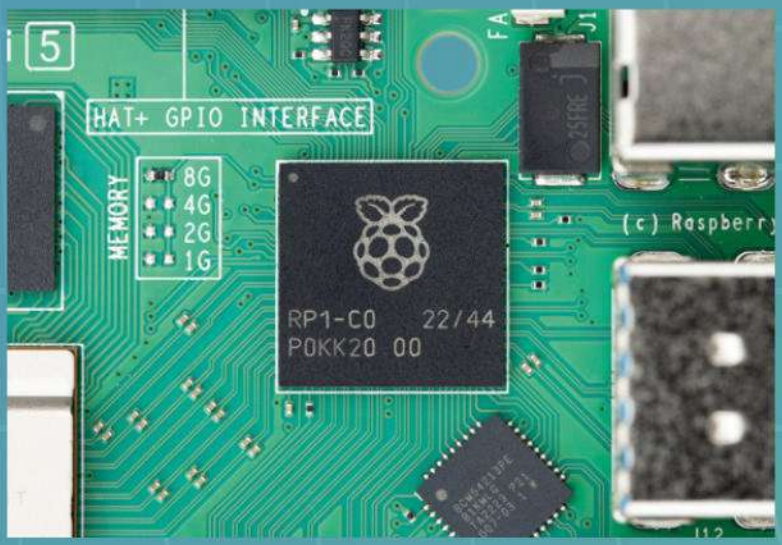
LEARNING TO THINK

The first bit of software we'll test is the motor drivers. Obviously, this will vary depending on exactly what hardware you opted for, but here is how to get the Adafruit control board running.

The board communicates with Raspberry Pi over a protocol known as I2C, so we need to enable support for this in the OS.



Above
Although this HAT doesn't fit on the new layout, we can route through the necessary connectors because the design is open



Above
All our input from the camera and output to the motors flows through the RP1 chip on the Raspberry Pi 5

Once you've got that, you can move on to the Python libraries. There's been a slight change in the way Python handles modules in the latest version of Raspberry Pi OS.

Previously, there was a slight problem. You could install packages via the operating system with apt, but also via the Python package manager pip. The two systems had different versions of the software available and could generally get in a mess with each other. Now, you can only install packages via apt. This should mean that you can't get into a system with incompatible packages.

However, there aren't as many packages available via apt as there are via pip. You also don't have as much control about the versions of things you install. The solution to this is virtual environments. This is basically a way of encapsulating a set of modules so you can use them in a particular project, but they don't affect the rest of the system. If you want to use them again in another project, you'll have to install them again in the virtual environment for that project.

```
mkdir robot
cd robot
python3 -m venv robot
```

By default, the virtual environment will exclude the modules that you have installed through apt. We don't want that, so we have to enable site packages. Open the **pyvenv.cfg** file in the directory with a text editor and change the line:

```
include-system-site-packages = false
```

...to:

```
include-system-site-packages = true
```

You should now find that inside your directory you have a directory called **bin**. This contains the executables for your project.

To use the virtual environment, you need to activate it with:

```
source ./bin/activate
```

If you close the terminal, you'll need to navigate to the directory and run this again in order to activate the virtual environment in the new terminal.

Inside the virtual environment, the Python and pip commands will be used under their local configurations. Anything you install with pip will only be available inside this virtual environment. If you want to install a package for multiple projects, you can either install it via apt (if it exists there), or reinstall it in each place.

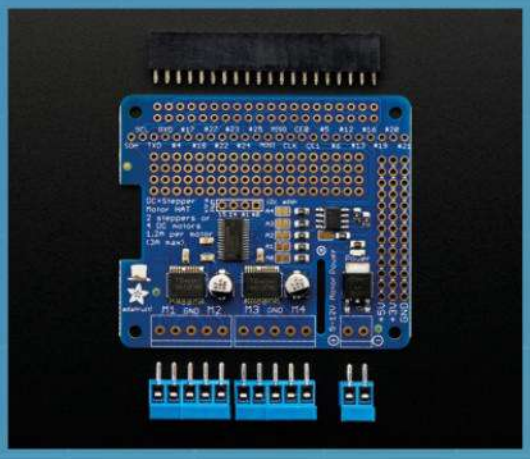
> **For this robot, we opted to place the cameras facing** diagonally out to give the robot a huge field of vision

With the virtual environment active, run the following to install the library for the motor HAT:

```
pip3 install adafruit-circuitpython-motorkit
```

Now run the following code. All the motors should come on.

```
from adafruit_motorkit import MotorKit
import time

kit = MotorKit()

max_speed = 0.5
motors_forward = [1, 1, -1, -1]
motors = [kit.motor1, kit.motor2, kit.motor3, kit.
motor4]

def all_to_speed(speed):
    for num, motor in enumerate(motors):
        motor.throttle = speed * motors_forward[num]
```



Right
This motor HAT is well supported and can power up to four motors

```
all_to_speed(max_speed)
time.sleep(3)
all_to_speed(-1*max_speed)
time.sleep(3)
all_to_speed(0)
```

This should spin all your motors forwards for three seconds then backwards. Depending on the wiring, the motors may or may not spin in the right direction. If they're spinning the wrong way, you can either change the wiring or change the value in the `motors_forwards` list to either 1 or -1.

If this throws any errors, then you'll need to fix them before continuing. The most likely problem is with I2C, so make sure it's enabled and that the wiring is correct.

ADDING INTELLIGENCE

Our motors now work. We'll get vision working before bringing the two back together. We're going to make our robot a search bot. It will use its cameras to detect particular objects and move towards them until it's got them.

The first step, then, is to get it to recognise things. We'll do this using the TensorFlow Lite neural network system. This takes a pre-trained model into which it feeds images. If the model sees something it's been trained to detect, then it outputs details of where the object is.

We're going to use the MobileNet model, which is trained on the Coco dataset. The latter is a collection of images that are labelled with 91 common items. Training on this dataset means the model should have a good chance of detecting one of these items in a broad range of situations. You can take a look at the dataset at **cocodataset.org/#explore**.

In our code, you can select one of these object types and the robot will try and find it. The actual search algorithm is incredibly primitive. If it sees the object on one camera, it will activate the motors on the opposite side and thereby turn towards it.

The full code for this is at **hsmag.cc/aibot**.

We'll need to install some dependencies for this to work. With the virtual environment active, run the following:

```
sudo apt install libatlas-base-dev

pip install tflite-runtime
pip install opencv-python
pip install pillow
pip install numpy
```

You'll also need the TensorFlow Lite model and dataset labels. You can download **coco_labels.txt** and **mobilenet_v2.tflite** from **hsmag.cc/ai-files**.

You can then run everything with:

```
python3 real_time_with_labels.py --model mobilenet_
v2.tflite --label coco_labels.txt
```

This should display on the screen two preview windows, and each one should be performing the detection separately.

You can control the particular thing that robot is looking for with the seeking variable:

```
seeking = "person"
```

The string in this variable should match one of the object types in **coco_labels.txt**.

The main control loop is:

```
def set_motors(left, right):
    left_speed = 0
    right_speed = 0
    if left:
        left_speed = max_speed * left_direction
    if right:
    right_speed = max_speed * right_direction

    kit.motor1.throttle = left_speed
    kit.motor2.throttle = left_speed
    kit.motor3.throttle = right_speed
    kit.motor4.throttle = right_speed

while True:

    buffer = picam0.capture_buffer("lores")
    grey = buffer[:stride * lowresSize[1]].
reshape((lowresSize[1], stride))
    found_left = InferenceTensorFlow(grey, args.
model, output_file,0, label_file)
```

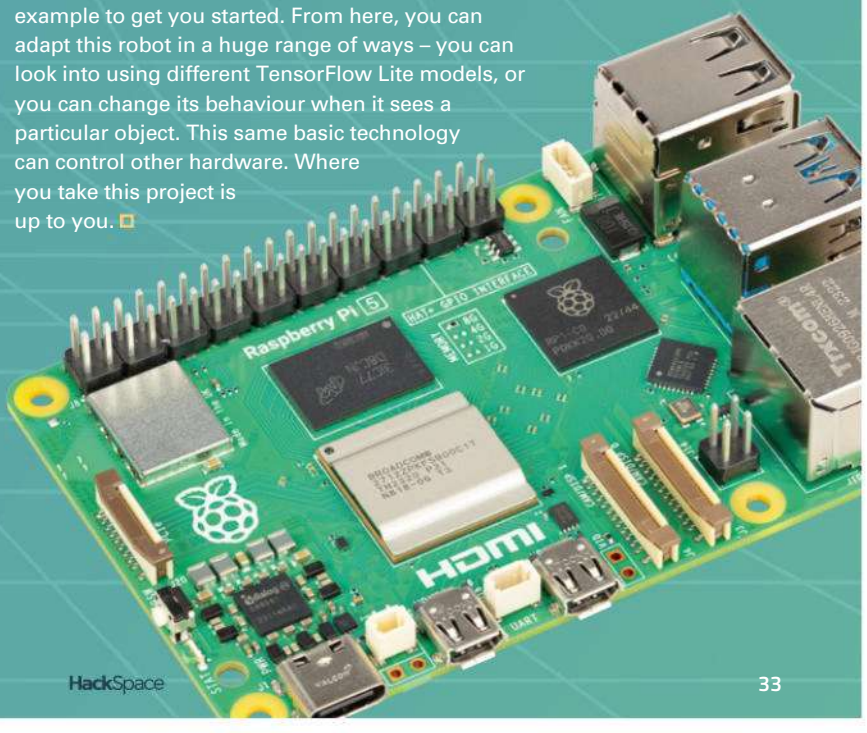
```
    buffer = picam1.capture_buffer("lores")
    grey = buffer[:stride * lowresSize[1]].
reshape((lowresSize[1], stride))
    found_right = InferenceTensorFlow(grey, args.
model, output_file,1, label_file)

    set_motors(found_left, found_right)
```

Picamera2 has the ability to return two images simultaneously, a high-resolution one and a low-resolution one. In this case, we're grabbing the low-res version to run the TensorFlow Lite model on. We're processing it to extract just the brightness component, and then passing it to the `InferenceTensorFlow` method.

```
def InferenceTensorFlow(image, model, output, camera,
label=None):
    global rectangles

    if label:
        labels = ReadLabelFile(label)
    else:
        labels = None

    interpreter = tflite.Interpreter(model_path=model,
num_threads=4)
    interpreter.allocate_tensors()

    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    height = input_details[0]['shape'][1]
    width = input_details[0]['shape'][2]
    floating_model = False
    if input_details[0]['dtype'] == np.float32:
        floating_model = True

    rgb = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
    initial_h, initial_w, channels = rgb.shape

    picture = cv2.resize(rgb, (width, height))

    input_data = np.expand_dims(picture, axis=0)
    if floating_model:
        input_data = (np.float32(input_data) - 127.5)
/ 127.5

    interpreter.set_tensor(input_details[0]['index'],
input_data)

    interpreter.invoke()

    detected_boxes = interpreter.get_tensor(output_
details[0]['index'])
    detected_classes = interpreter.get_tensor(output_
details[1]['index'])
    detected_scores = interpreter.get_tensor(output_
details[2]['index'])
    num_boxes = interpreter.get_tensor(output_
details[3]['index'])

    for i in range(int(num_boxes)):
        top, left, bottom, right = detected_boxes[0]
[i]
        classId = int(detected_classes[0][i])
        score = detected_scores[0][i]
        if score > 0.5:
            if (labels[classId] == seeking):
                print("found on camera ", camera)
                return True
    return False
```

After setting up the data in the correct format, this runs the TensorFlow model which returns a list of objects it's found and details about them (including a bounding box). We go through this data looking for any that has a score over 0.5 (the score is how confident TensorFlow is in the object), and a class that matches the `seeking` variable. If something matches this, it returns `True`, otherwise it returns `False`, and we use these return values to determine which motors to engage.

Our robot is in some ways very simple: there are four motors that are either switched on or off depending on whether a person is seen on a particular camera. On the other hand, it's really complex – it's using a neural network to identify the objects in an image. We're just using it as an example to get you started. From here, you can adapt this robot in a huge range of ways – you can look into using different TensorFlow Lite models, or you can change its behaviour when it sees a particular object. This same basic technology can control other hardware. Where you take this project is up to you.



Below
The Raspberry Pi 5 has enough power to run object recognition on both cameras at the same time

SUBSCRIBE TODAY

GET SIX ISSUES FOR JUST:

£30 UK / **€43** EU / **$43** USA & Canada

GUARANTEED

RASPBERRY PI 5 AVAILABLE NOW

for all subscribers

SUBSCRIBER BENEFITS:

- **Get every issue of HackSpace magazine delivered to your door**
- Beat the crowds with guaranteed Raspberry Pi 5 stock for subscribers
- Early access to the PDF edition
- **Get a free Raspberry Pi Pico W**

hsmag.cc/subscribe

Subscribers will get a voucher giving them the chance to purchase one Raspberry Pi 5 from reserved stock at The Pi Hut (thepihut.com) for full retail price. Reserved stock means that these will be available even if they are out of stock for general purchase.

FEATURE

HOW I MADE

By Turi Scandurra

JUKEPHONE

Want some music? Just dial-a-tune

As an electronics tinkerer, I get a thrill from perusing the shelves of local charity shops in search of old gadgets I can creatively revamp. I look for things to take apart to see their inner workings, and uncover the system of mechanical and electronic parts that make them function. On one such quest, I came across a dismissed landline telephone with chunky keys, which was practically begging to be hacked into something else. The moment I saw it, ideas started swirling in my head about how I could give it a new life.

I initially focused on the keypad alone, thinking it could become a comically oversized numeric pad for a laptop (I might still do this). I would have then been left with a decent, spare speaker and a microphone. But I knew that with the right alterations, this retro relic could sing a new tune. So, in a musical twist, I came up with the idea of a personal jukebox, encased into the telephone, that would let me play tunes by dialling out their track number.

I started by disassembling the device and inspecting it to figure out how to incorporate as many original components as possible into my new design.

What I used

- Landline telephone (I'm afraid a rotary dial phone won't do it)
- Raspberry Pi Pico
- DFPlayer Mini (or MP3-TF-16P clone) – MH2024K-24K, MH2024K-16SS and many more chips are supported
- microSD card – 8GB or more is recommended
- TP4056 battery charger module
- JST plugs
- 18650 or equivalent lithium battery
- 1000 µF electrolytic capacitor
- 2 × 1 kΩ resistor
- 3.5 mm audio socket

The keypad, with its playful large keys, was surely the main hardware feature and had to stay in its place. The little status LED and the piezoelectric buzzer, which I carefully desoldered from the PCB, seemed perfect to serve as feedback indicators. The telephone had a toggle switch on the back to set the ringer volume. I could have wired it to act as a power switch, but then I decided to lose it and use the spring-loaded switch underneath the handset instead. I believe this change was a clear steer towards a more accessible, human-centric design. There is some beauty in the way you answered a call on a landline telephone – you would just lift the handset and listen. With no buttons to press, it was just the natural gesture of bringing the speaker to your ear. That's the right level of simplicity that I wanted for the final users of my new object. And after all, listening to music through the handset speaker is what gives this music player its quirky character.

Since the original PCB was inevitably going to become e-waste, my creation needed a new brain. The microcontroller I use most often these days is the RP2040, specifically in the form of a Raspberry Pi Pico. It's powerful but also energy-efficient, offers plenty of GPIO pins, and its price is very affordable.

Above
Raspberry Pi Pico provides a powerful and affordable brain

FEATURE

Above
The phone's battery is recharged via USB-C

> *" I USED FOUR C LIBRARIES"*

While the Pico can be programmed to produce audio output via pulse-width modulation or I2S – like I did with some of my previous projects – its capabilities are not quite right to get high-quality music playback. So, I picked a separate music player module, the popular DFPlayer Mini. It can play audio files from a microSD card, has a built-in amplifier to drive a small speaker, and can be controlled digitally via UART communication.

Armed with my soldering iron, I first rewired the telephone's keypad through a ribbon cable and to a perfboard on which I had soldered two pin header sockets for the Pico. The connection between the keypad and the Pico GPIO pins allowed me to start writing code and let the controller detect key presses.

For this project, I used four C libraries that I had already written and one created by another developer that I ported to the Pico. I like to break down my code as much as possible into a modular structure that lets me reuse components between projects, and every project is an opportunity to write new libraries.

C LIBRARIES

The most recent one is RP2040-DFPlayer, which implements the UART communication protocol to send instructions to the MP3 player and poll its status. Without it, the features of the Jukephone would have been limited to only basic actions like starting the playback of the first track, dialling the volume up or down, and skipping to the next track. I took time to study the datasheet of the player and unlocked features like equaliser presets, playback modes, and querying of playback status.

RP2040-Keypad-Matrix is another crucial component of this project's software. It alternates write/read cycles across the rows and columns of the matrix to poll it for changes, and discern between short and long key presses.

Even though the buttons produce a tactile feedback on their own, I made it so

that the little LED blinks shortly after each key is pressed, together with a short beep emitted by the phone buzzer. For that, I used RP2040-PWM-Tone, my tone generation and melody player library for Raspberry Pi Pico. I initially used it to play a start-up melody, but finally settled for a quiet start-up because I feared a jingle would get annoying in the long run. When designing products, it's tempting to add features just because the tech allows you to, but 'possible' does not mean 'necessary'. Ultimately, this is an open-source project, so adding the melody back would require just one line of code, since a few sample melodies come bundled with the library.

A nice little utility that I add to all my machines is provided by the library RP2040-Battery-Check, which, as the name suggests, periodically checks the battery voltage and rapidly flashes the LED when it's time to recharge it.

The last library I included is RP2040-Button (the one I ported from another developer's work), which is needed to detect presses of one lone key that is not wired with the rest of the matrix.

The main software logic is pretty straightforward. Key presses are debounced, numbers concatenated and clamped between 1 and 999. When a valid track number is entered, after a short timeout the MP3 player is sent a command with the ID of the new track to play. The chip on the MP3 player is able to pick a random track to play on its own, but I added my own playlist randomisation function on the Pico, which means that tracks do not repeat until the whole playlist has played entirely. However, the Jukephone I set up contains over two days' worth of music, so I don't think it will happen often.

UART communication between the Pico and DFPlayer uses just two wires for TX and RX, one of which is filtered with a 1 kΩ resistor to reduce noise.

POWER PARTICULARS

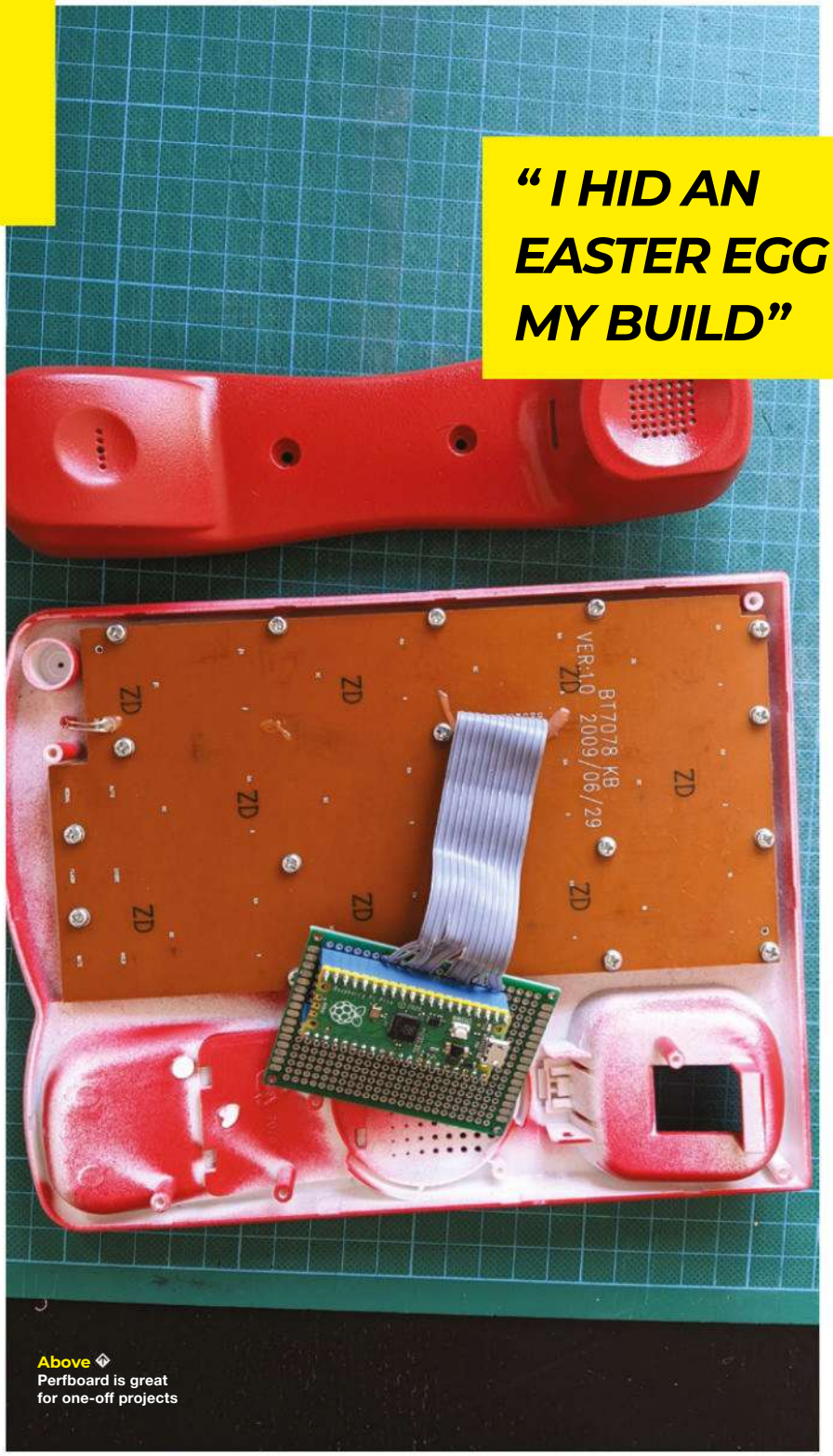
The output pins of the MP3 player go straight to the speaker inside the handset. There's also a 3.5 mm mini-jack socket, connected to the player's built-in DAC, so you could plug headphones or external speakers into the back of the Jukephone. Sound quality is much higher through this output and, unlike the handset, it's a stereo output.

The whole project is powered by a 3.7 V lithium battery, recharged by a TP4056 module via USB-C. This little module is a staple for all my portable designs as it also provides overcharging and undercharging protection, for prolonged battery life. I can say I treat my batteries fairly well because I never heard them complaining.

Below
I'm pleased with the finish on the red paint

FEATURE

JUKE



Above Perfboard is great for one-off projects

"I HID AN EASTER EGG IN MY BUILD"

It might have been overkill here, but I've taken the habit to always add a decoupling capacitor of around 1000µF just after the power stage to help stabilize the voltage and squash any ripples that might pop into my circuits. It's worth noting that technically the charging module could simultaneously provide energy to the battery and the rest of the circuit, but doing so is not recommended as it interferes with the charge cut-off, increasing the risk of overcharge.

Opting for no changes to the original plastic housing meant that I had to repurpose the openings for two toggle switches and for the RJ11 socket that once connected the telephone to the wall. They became natural slots to access the microSD card, the USB-C charger, and the headphone jack.

FINAL ASSEMBLY

My perfboard – holding the Pico, the player, and the few discrete components – was perforated further in order to open two mounting holes and secure it to the housing.

I 3D-printed a custom mount for the TP4056, which locks nicely around the housing wall and a standoff, while also raising the module to its perfect placement and angle. The lithium battery, connected to the charger via a pair of JST plugs to make it swappable, was secured to the housing floor with a dab of hot glue.

The telephone's exterior went under a DIY makeover with a few coats of spray paint. And no other colour screams 'touch me' like shiny red! I then labelled the keypad buttons with metallic purple paint markers. The coiled handset cord is the only part I needed to replace, as painting over the old one was not going to offer a durable finish.

I loaded the microSD card with 999 MP3 files, organised so that there's one hundred per genre (except the first 99). Specific tracks can be invoked by typing their number on the keypad. I replaced the old directory card under the handset with a printout of the genres available.

PHONE

Above
Big numbers and bright red: this phone wants to be noticed

The six additional keys below the large numeric keypad were programmed to perform useful actions, like increasing and decreasing the volume, pausing and resuming playback, restarting the current track, toggling repeat mode for the current track, playing a random track, and rotating between five sound equalisation presets.

In thinking of a future iteration of this project, my mind goes to the possibilities of an internet-enabled device. While still retaining the central feature of track selection by numeric input, it could enable a Wi-Fi back-end to customise the playlist with web resources to stream.

I hid an Easter egg in my build. Dialling a specific number triggers a voice message that says: 'Thank you for calling the Jukephone Helpline. All our operators are busy at the moment, please try again later.'

I can imagine the small person who will receive my first Jukephone as a gift getting ready for bed, tapping out children's tunes until she gets sleepy. Hopefully it will stay functional until she's old enough to appreciate the rest of the playlist.

Finding an old gadget a new home rather than the landfill is rewarding both creatively and environmentally. I can feel my Jukephone evoking a sense of nostalgia for relics of the past, but in the end it represents the joy of second chances. And even though it's not really a telephone any more, it still does phone home in a certain way.

LINKS

- turiscandurra.com/circuits
- github.com/TuriSc/Jukephone


FROM THE MAKERS OF The MagPi THE OFFICIAL RASPBERRY PI MAGAZINE
NEW 2022 UPDATE
RETRO GAMING WITH RASPBERRY PI
2ND EDITION
164 PAGES OF VIDEO GAME PROJECTS
+ BUILD AN ARCADE MACHINE
PLAY & CODE GAMES!

HackSpace magazine meets...

Matt Venn

Chip design: the next frontier of open-source hardware

We're familiar with the idea of open-source hardware: designs that a motivated user can download, modify, and rebuild for themselves. But the idea of designing your own chip is surely out of bounds for mere individuals. The Raspberry Pi Foundation spent in the region of $10 million to develop the RP1 chip that sits on the Raspberry Pi 5, for instance; Apple, using a more modern production technique, will have spent many times more than that. Thicknesses of functional parts are measured in nanometres, and are getting smaller with every generation of chip manufacturing process. The Raspberry Pi 4, for example, used an Arm Cortex-A72 chip, which used a 16nm process; right now, the world's leading chip manufacturer, TSMC, is developing a 1.5nm process. The scale of the investment required is mind-boggling.

Luckily for us, there is a way that we can get involved. Matt Venn, with his Zero to ASIC course (**zerotoasiccourse.com**) has partnered with Tiny Tapeout, a service that enables users to buy a part of a chip wafer on which they can have engraved their own chip design. It's beyond the ability of most to create a fully functional CPU, but we can instead make a chip that does one thing, and one thing only: an application-specific integrated circuit, or ASIC. Welcome then, to the mind-blowing world of homebrew chip design.


ASIA

HACKSPACE Zero to ASIC then: what's it all about?

MATT VENN So it's called the Zero to ASIC course; the idea is that you can take the course and go from zero chip design, or digital design or ASIC knowledge, and at the end of it, do a takeout and get a chip.

HS Literally zero knowledge?

MV Well, a bit of prior programming experience is useful; you don't do any programming in the course, but it just means that you're familiar with an editor – you're familiar with writing code-like stuff. And I also say that a bit of Linux experience is useful, because all the tools are mostly Linux-focused. If you're hitting the command line for the first time, that can be a big jump. We have a support forum. We use Discord for that. And we have a section of that for people who need Linux support. Questions like 'What the hell is a command line anyway?'

I normally include the top-level instructions in the course material, and one of the things that I have done is I always record myself completing the tasks that I'm setting. So the course is made up of labs, which are practical experiments. And I record myself solving the challenge, doing the practical experiment, and that's a way for people to get back on track if they get truly lost.

HS How long does it normally take someone to go from zero to designing a completed chip?

MV It really depends on how much time someone has. It takes maybe 30 or 40 hours of study and practice to get from one end to the other. Some people spread it out over six months or a year. And some people do it in one week.

If you're a student and it's your summer holiday, and you're really interested, you can just smash through it really quickly. If you've got a full-time job and a family, then obviously it's going to take you longer.

The course is fully designed to be asynchronous. I always wanted it to be able to scale beyond the kind of synchronous workshops that I used to run regularly at hackspaces and makerspaces.

HS On your website, in the FAQs, you talk about the fact that you use open-source tools. As I understand it, microchips are a multi-million billion trillion dollar industry – so how on earth is it possible to design something like a microchip with open-source tools?

MV Let's just go back in time and think, when Linux first came out, people were paying a lot of money for operating systems. The idea that you could have a free, open-source operating system was ridiculous. And the same thing happened with MySQL, the database. And the same thing happened with the GCC compiler. And then Arduinos – before Arduino, you had to spend $200 or $300 on a programmer, and pay for the licence of the software, and so forth. And then, when the Arduino came along, you just needed to buy a $30 board and plug it into your computer, and you were off.

> "Instead of designing a chip that's a $100,000 endeavour, make it a $100 endeavour"

So, this is the beginning of the journey of open-source chip design. You can pick examples of stuff that we're doing that is pretty impressive, like RISC-V cores with built-in 5GHz radio transceivers and stuff like this. But to be honest, the majority of the projects are kind of experimental, or people trying things out.

Most of the people involved are not professional chip designers. I was looking at a submission this morning, which was a re-creation of an old-school sound chip. Someone's had the idea to recreate a chip that you can't buy off the shelf, or maybe you could have bought off the shelf and you can't any longer.

There's also a rule of thumb, which is that an FPGA [a field-programmable gate array – an integrated circuit that can be reconfigured to fit different use cases] will generally perform ten times better than a CPU, and an ASIC will perform ten times better than an FPGA.

If you've got a given task that you want to complete, then an ASIC will be 100 times faster than a CPU. That's why when Bitcoin was all the rage, the only way to really be involved was to buy ASIC miners.

We're using a 20-year-old [manufacturing process], so we lose that 10 times performance gain; probably, if you're very good and experienced, you might be able to get equivalency on a CPU if you knew what you were doing. So we can't really compete with, say, the RP1, the chip that just came out on a 22-nanometre process, where people are buying in IP blocks from here and there that are very performant already.

But you can do weird experiments that you wouldn't dare to do if you were spending a million pounds on every tapeout. So really, the idea is to get more people into it, to open the door. Instead of designing a chip that's a $100,000 endeavour, make it a $100 endeavour. And see what happens. Maybe we'll see the same revolution that we saw with Linux, MySQL, GCC, and Arduino.

HS Are we reaching a sort of natural user limit with Moore's Law, that transistors on CPUs just physically can't get any smaller, so to increase performance, you have to build chips dedicated solely to one task rather than using a general CPU?

MV There are quite a few different strands. There's Moore's Law coming to an end. Then there's... the West has experienced supply chain shocks during Covid, that have awakened people to the fact that the

semiconductor supply chain is the longest and most complicated in the world, and almost every high-value industry sector is dependent on semiconductors at some level. So, if you can't get computer chips or servers, then you probably can't run your business anymore.

Like, what happened with the car manufacturers – they just had to shut down their factories because they couldn't get chips. So there's Moore's Law, and there's wanting to be independent and that independence.

The EU and the US have put in $100 billion together to invest in the sector. But building factories and building capabilities is no good if you don't have the people.

And in the EU Chips Act, they're saying that there's going to be something like a 100,000 people shortfall in the next ten years. So where do those people come from?

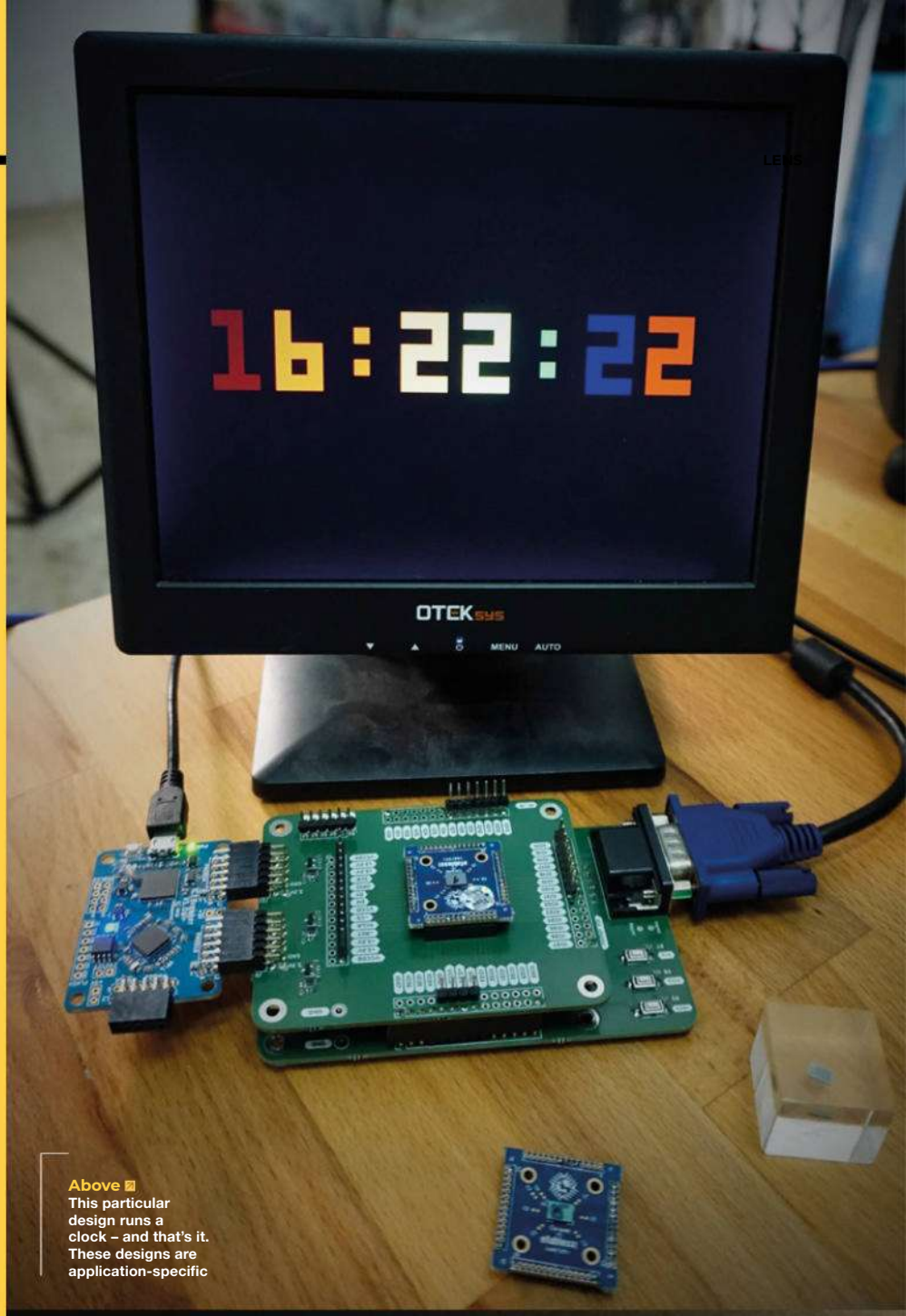
Well, maybe they did some basic digital design stuff, or some basic ASIC stuff when they were a teenager, rather than at PhD level, which is when you normally get to do your first tapeout.

People have been saying Moore's Law is ending every year for the past 50 years, but there's no denying that the amount of money and time that goes in to get the same gains that have been seen over the last 20 years is now more and more exponentially higher.

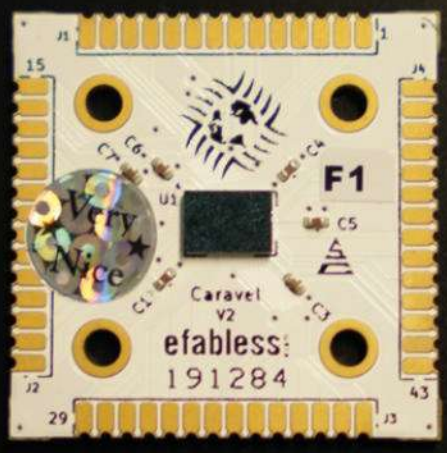
Like the amount of money that was spent on developing extreme ultraviolet lithography [a new process that enables even smaller transistors] at TSMC, and they're still getting only a 50% yield or something on that process, they're really struggling with it; it's not at all like an old node, like the one we use.

And you can go up: all our chips are really flat. All the complexity is in this very thin layer. But if you go up, you have a real problem with getting power in and heat out.

Processors are already constricted with power in and heat out. If you build it 3D, you're gonna get huge processing power, but it's not actually possible to do because now you've got terawatts of power to get out of there.



Above
This particular design runs a clock – and that's it. These designs are application-specific



Below
Matt's helped students, professionals and curious tech geeks design their own chips

Above
Matt's been learning and teaching ASIC for around three years now, and has the hat to prove it

You can make other more exotic types of transistors, but you can't keep on shrinking them. Because we're now at the atomic limits. So people are looking around for other ways that are more cost-effective to get more power for your dollar. And one way that you can do that is with specialisation. For example, YouTube has racks of ASICs that just compress video. It's worth their while making a custom chip just for that one thing.

A general-purpose computer is general purpose, but that means it's pretty poor at everything basically. That's why you have graphics cards: because we all depend on graphics, and people like playing 3D games or using 3D for CAD or for chip design or whatever. And that needs acceleration; the CPU can't do it. So you have a separate accelerator. Everyone needs a big screen; we're all visually focused. So it makes sense that one of the first things that was accelerated was screens and graphics. But if you're doing a lot of encryption or a lot of video encoding or a lot of simulation of climate change or training HIs or whatever, then it makes no sense to use a general-purpose computer, because you can get ten times the performance with something specialist.

HS That feels like going backwards. It feels like going back towards the days of Alan Turing building a huge machine to do one thing.

MV Back then, general-purpose computing was a massive breakthrough because you could build this one thing and then do everything with it. But we've lived with general-purpose computing for like 50 years now. We're kind of at the limits of that. And people still want more performance.

You could also write your computer programs more efficiently: a tonne of web apps and software as a service apps are written in JavaScript or whatever, and that's running in the browser, which is running on the OS, which is running on an abstraction layer, which is running on a kernel, which is running on the CPU. You can see how you lose efficiency at each of those levels. So you can write closer to the metal – which is harder and takes longer time and takes longer to develop, but you'll get more performance – or you actually build different metal.

HS It's not the same thing, but Raspberry Pi is at the stage where they're designing the chip around the board, rather than designing the board around a chip.

MV In a way, what Raspberry Pi is doing now is chiplet design, but instead of using a silicon interposer they're using a PCB. It's very clever, because they're able to take advantage of like 20 nanometres or lower for their core from Broadcom, an off-the-shelf component made in huge volumes, and then the thing that makes the Raspberry Pi special and easy to use, and where all the documentation is built around, and all the examples and all that glorious ecosystem that is the killer app for Raspberry Pi – that's in your own custom hardware, which can be at a bigger, cheaper process, and it can be tiny, and you can get tens of thousands from a wafer. It makes a lot of sense.

HS To my uneducated eye, the process of making an ASIC looks a little like doing a PCB design, where you do a digital design and then it goes through an additional step to translate that into physical reality. Is that right?

MV That's maybe half the story: you can roughly divide chip design into two threads. One is digital design for building a CPU or a logic chip or something like that. And the other side is things that interface with the real world, like analogue-to-digital converters, or radios or PWM drivers or GPIO pins, or temperature sensors.

All of that stuff is more broadly divided into analogue and digital. Analogue looks quite like PCB design, and it's still a very manual process with people drawing out shapes.

The digital side of things is one layer more abstracted. So we have a library of what's called standard cells. And they do defined logic functions that have already been designed by somebody else usually, like an adder or a flip-flop or an AND gate or an XOR gate or combinations of gates that are useful for things.

When you're designing digital circuits, it's still possible to grab loads of these and then wire them all together. But it would be more normal for a person to say, 'I want a register with 10 bits of information in it, and then another register with 20 bits of information in it, and I want to add them together, take the output and put it back into the first register'. And you would do that by writing in what's called a hardware description language, where you describe the hardware you want in words rather than by drawing pictures. So it's a level higher in abstraction.

HS This may be a silly question, but once you've got the chip that you've designed, you can't just then run Python on it can you?

MV Python is quite a long way away from the metal.

Yeah. The whole world of the C programming language is a good way to explain it. With C, you've got a linker, and a memory map, and some assembly, and then a compiler – all those things come together so that you could write some assembly. And it would run on that processor, because it would know what the registers are, what the capabilities of the CPU are, what the memory map is, where the reset address is, and all this kind of stuff, and then you might be able to make a binary that you could load into the CPU and it would execute it. And then, once you have that, then you write C with it, and once you have C, you write Python with it.

HS A lot of sensor modules that home makers use are affordable now because they're a few generations behind the cutting edge. Is that the case with what you're doing?

MV All the chips I design are using what's called a process development kit [PDK], which is like a secret database that the factory creates when they set up their tools.

You've got to remember that these factories cost billions of dollars. I think the latest TSMC one for their N3 process, one of the very new latest ones, cost $18 billion for the factory. But at the other end of the spectrum, you're still talking of hundreds of millions, because you still need a very clean room because even one speck of dust is going to ruin a chip (because specks of dust are way bigger than the features that you're making). And all of the equipment is really expensive. I visited a factory called IHP In Germany, they have a 130-nanometre process. And the lithography machine that flashes a light through a pattern onto this sensitised wafer was, I think, between $10 and $20 million for that one machine.

That's one of the more expensive machines, but you need like eight or ten machines minimum to run a chip line, as well as the clean room and maybe 50 people who are all specialised, even for an old process like Sky130. And because it's so much bigger, it's *only* 130 nanometres, that means that when you're making your masks that you shine the light through to do the features on the very smallest features have like the minimum thickness is 130 nanometres on a 130-nanometre process.

A 3-nanometre process doesn't have anything on it that's 3 nanometres. It's confusing, but in the old days, the numbers still had a relation to the product. So with a 130-nanometre process, that means you need to make wires that are 130 nanometres. So you send this chunk of glass off with a chromium top to another factory that will just put the features in. And they might use something like an electron beam, or a focused ion beam to cut away the part of this thin sheet of chromium on the glass. That's usually four times bigger than the features that you want. So that means you're cutting lines that are maybe 600 nanometres, so half a micron, so a bit more imaginable. And then through the optics system, when you flush that light, it goes down four times smaller and exposes onto the wafer.

These chips are made up of maybe 100 layers, which means you need 100 masks, but only the ones right at the bottom are the ones with the very fine detail; the ones at the top are made on a much bigger scale, so those are cheaper to make because you have less detail on them and you can use older machines.

So all those 100 pieces of glass with 100 patterns on to make your chip on Sky130 would cost $200,000. But once you have them in the factory, and it's all going in its automatic process, you can be stamping out wafers with 2–3000 chips on it, and they're just coming out of the end of the line, boom, boom, boom, boom, so that's where you get the volume, that's where you get the things that are actually affordable to buy at the end, because you've got that mass production process.

> "Every undergraduate studying microelectronics should do a tapeout"

If you're talking about the masks that were used for the RP1, they probably cost a million dollars for the mask set. So that's another reason why it's more accessible to use an older process, because the masks are cheaper.

And then the other thing [that keeps the price down] is that on each wafer there's a square that has about 40 little smaller squares, and that gets repeated over the whole wafer.

So then those 40 people split the 200 grand between them, they only pay ten grand each. And then what I do is I split that one chip into another 400. And that's why I can charge you $100 to do a tapeout.

HS So when you get the thing back that I've paid $100 for, it's not just my design that I'm getting?

MV Exactly. Your design is one of these little designs, and then that whole thing is inside this square of silicon, along with another 300 designs, and then you use this bank of switches on this PCB here to set which design you want to be active.

HS You mentioned TSMC before. Is that where the chips get made?

MV We don't send them to TSMC, because we don't have the volume. They're only really interested in big deals. Or local education – if you're in Taiwan and a you're a student, you get free tapeouts. It's normal for people to do their first tapeout at PhD level. And one thing I'm really working on with my projects is that every undergraduate studying microelectronics should do a tapeout. We're also working with high schools.

You know when the UK adopted the stance that everyone has to learn how to program a computer? There is some sense in that, but not everyone needs to know how to program a computer, and forcing people to do it if they don't want to is maybe counterproductive. But making it available so you can choose to do it if you're interested in it, I definitely support that. So it could be nothing like that: if you're a motivated high school student, your final project could be to do a tapeout and make your own chip. Why not?

HS TSMC is in Taiwan, which is only a hop and a skip from China. Have you found that there's been an increase in interest since recent geopolitical events made Taiwan's proximity to China more obvious?

MV That, and the supply chain shocks were the reason for the EU and the US Chips Act. And that $50 billion each is being invested in lots of different places; probably 90% of it will be spent on equipment or buildings. But 10% of a billion is still a lot of money. And they want to build centres of excellence for training around Europe. And that's happening in the US as well.

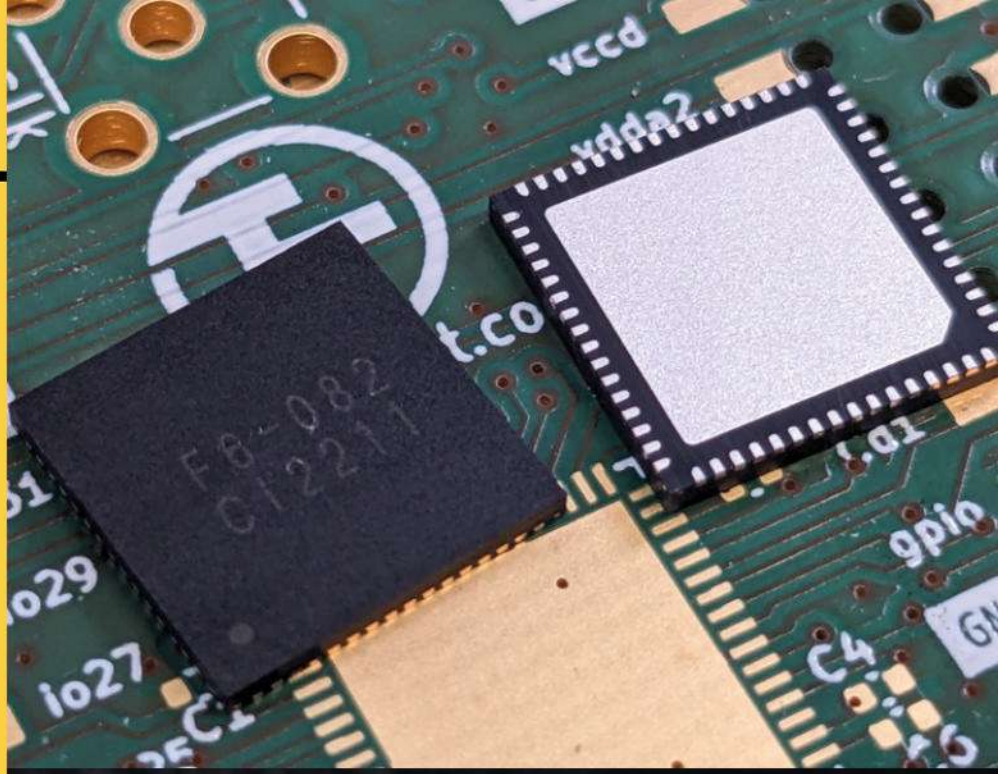
I'm collaborating with universities in Europe and in the States with training and with the TinyTapeout project to help people do the practical side of things. I mean, I'm a very practical learner; I wouldn't have made a chip design course if it wasn't possible to actually end up with a chip in your hand. And for other people who learn in a similar way, or who also value the importance of making physical things, I think it's important.

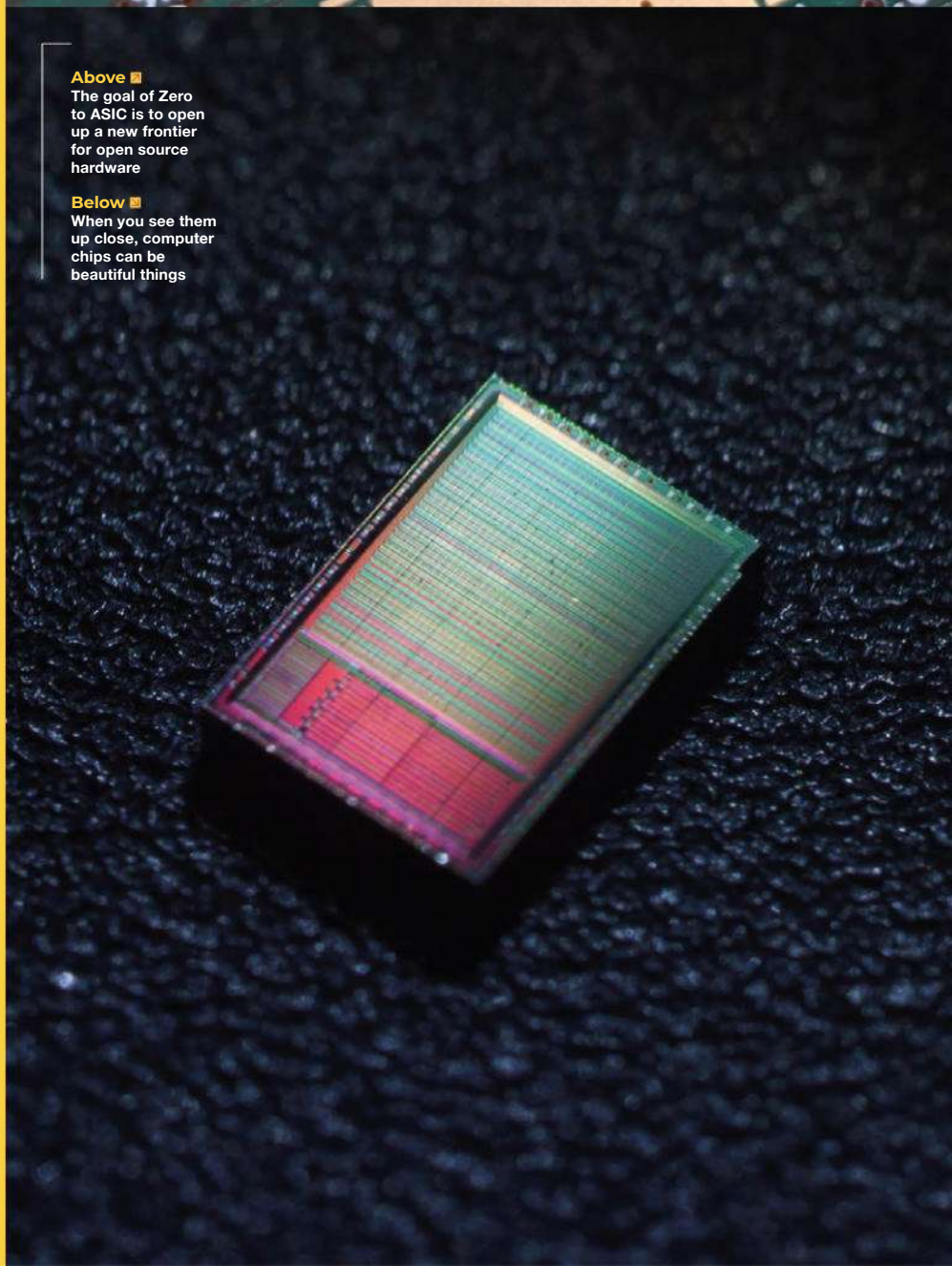
It's happening, definitely. And some small amount of money is being invested in the open source side of things as a kind of a side bet maybe. The big boys, like Synopsys, Cadence, and Siemens, they dominate the chip design world for good reason, because between the three of them, they've probably invested a trillion dollars in R&D over the last 40 years.

So of course, they're way ahead of any open source effort. But open source makes perfect sense for training and education, because a licence for one of these tools costs $100,000 – how can you afford to let your students use it? Universities make deals with these big companies, so they get it cheap, but then only the big fancy universities get to benefit from that, and a small university in Vietnam or Ecuador, they can't get access to the tools and they can't teach chip design. Open-source tools makes a tonne of sense.

You see people doing things that we've never done before with PCBs. People using PCBs for things that they were never designed for, like mechanical enclosures, or for decoration, or all kinds of stuff, and some of that does feed back into industry. I think that changing the mentality of chip design from being a massive endeavour with hundreds of people and tens of millions of dollars, and you can't make any mistakes. That working environment has an effect on people. And it limits creativity and out-of-the-box thinking, because you're too scared to make a mistake.

So having very cheap access to silicon and experimental tools that you can even write yourself. We have designs that are now coming from people's home-designed tools. Who knows where the next chip design startup is going to come from?





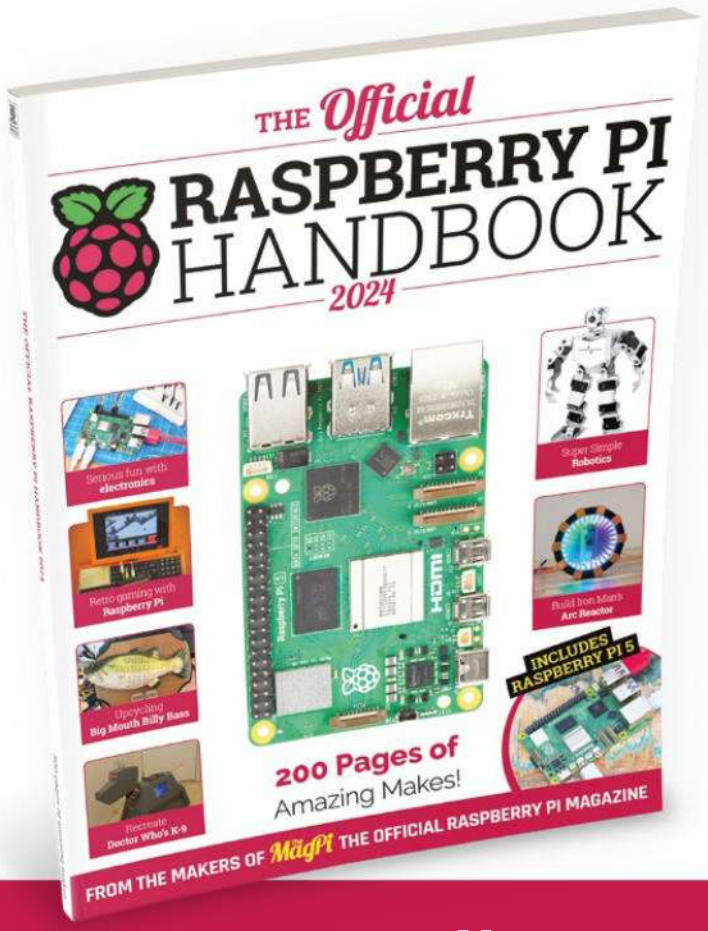
Above
The goal of Zero to ASIC is to open up a new frontier for open source hardware

Below
When you see them up close, computer chips can be beautiful things

THE Official RASPBERRY PI HANDBOOK 2024

200 PAGES OF RASPBERRY PI

- QuickStart guide to setting up your Raspberry Pi computer
- Updated with Raspberry Pi Pico and all the latest kit
- The very best projects built by your Raspberry Pi community
- Discover incredible kit and tutorials for your projects



Buy online: magpi.cc/store

HackSpace
TECHNOLOGY IN YOUR HANDS

FORGE

HACK | MAKE | BUILD | CREATE

Improve your skills, learn something new, or just have fun tinkering – we hope you enjoy these hand-picked projects

TUTORIAL

PART 01

Pico modular

How practical is a cheap, hackable modular synthesizer?

Ben Everard

Ben's house is slowly being taken over by 3D printers. He plans to solve this by printing an extension, once he gets enough printers.

A **modular synth isn't a singular thing: it's more a concept.** It's an electronic instrument that's built up out of discrete parts (or modules) that can plug together and communicate using a standard protocol. Typically, the standard protocol is an analogue voltage known as control voltage (or CV), and is transferred around using jack-to-jack cables known as 'patch cables'. Using these, a set of modules can be combined in different ways to create different sounds. Modules usually come from many different suppliers, and there's a huge range to choose from, meaning that there is no standard modular setup (though there are certainly some common parts that many modular synths have).

WHY PICO?

There are a lot of different microcontrollers we could have chosen for this, and they have different pros and cons.

Raspberry Pico has three analogue inputs. This isn't as many as some, but it's a few.

With Pico, it's easy to control hardware with accurate timing across a range of interfaces. It's powerful enough for our uses and, significantly, it's cheap enough that we can throw a lot of them at the problem without increasing the cost too much.

It's the price and feature set of Pico that really makes this project possible in its current form. Throwing this number of almost any other moderately powerful microcontrollers at the project would mean that it got too expensive to work.

They often (but by no means always) take input from other systems such as MIDI keyboards.

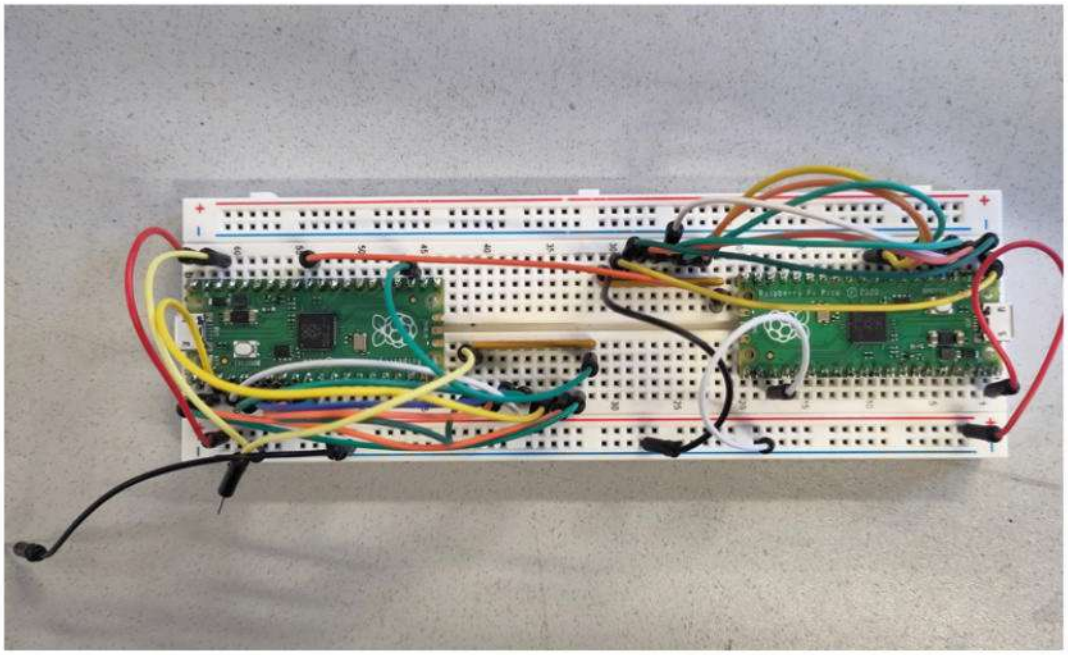
Modular synths are a fascinating and adaptable way of building electronic instruments. However, they are big and expensive.

Part of the reason that analogue synths are so big and expensive is that they tend to be focused on high quality. They're robustly made and produce high-quality sounds. This is a good thing, but it does mean that it can be an expensive world that's hard to get into.

We've decided to go back to basics and reimagine what a modular synth could look like if it were created in 2023. We want to keep the intuitive voltage-based system, but scale it back so it's easy to work with 3V microcontrollers, specifically Raspberry Pi Pico. In this series, we'll build it up module by module until we've got a system that we're happy with.

This is not a particularly efficient way of building a system. Pico can do many more of the tasks than we're giving it. Also, if you are going to transfer data between microcontrollers, using analogue is, at best, antiquated. There are plenty of much more modern digital protocols that we could reach for. However, music has never been about practicalities. It's about aesthetics and how an instrument makes you feel.

By making the communication analogue, we're leaving the door open to communicating with other modular synth hardware. Our simple setup isn't designed to work this way at the moment, but it's something that we'll tackle in the future. Analogue voltage communication is also intuitive to understand, and we might look to add some purely analogue modules in the future.



Left
The easiest way of experimenting with Pico modular is on a breadboard

THE ROAD AHEAD

We have a target of building a playable modular system for under £50, and a more interesting system for around £100. We will probably create more modules than this, but it's up to the reader which ones they want to create. To put it another way, we want to make a modular synth for less than the price of a single typical module. That is ambitious but not impossible (depending on what you want to call a playable synth).

> We want to make a modular synth for less than the **price of a single typical module**

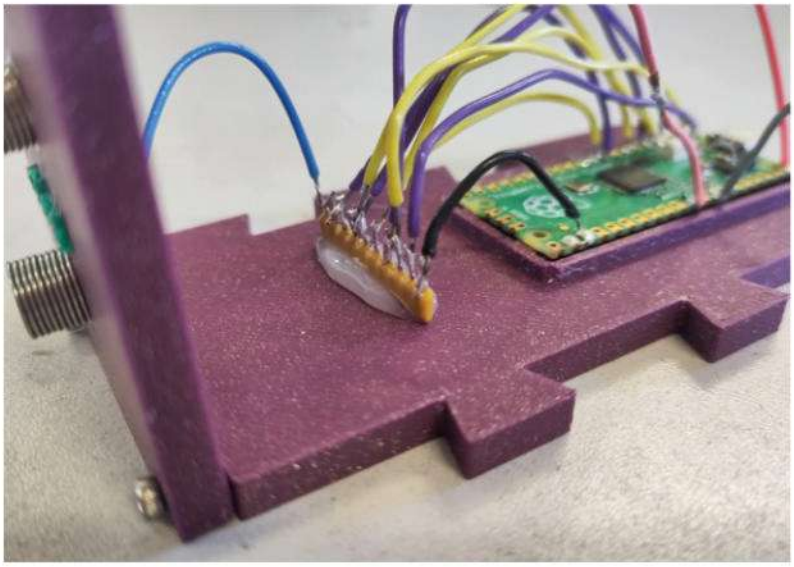
To do this, we need to throw away lots of things that are good practice, but not 100% necessary. This includes protections for the inputs, and various bits to filter and improve the sound, and the range of supported voltages. We will, in a future article, look at the impact of adding these back in to make our synth more compatible with a more traditional synth, and look at what the pay-off is in terms of price and quality.

Modular synths are usually (but not always) built around the Eurorack standard for physical setup and power. We're going to abandon this as it's too big and expensive for our needs. For physical setup, we're going to work with 3D-printed enclosures (though you could easily replicate them with hand tools if you don't have access to a 3D printer). For power, we're going to start with a standard 0–5 V range for powering the modules, and 0–3.3 V for analogue communication.

ANALOGUE IMPERFECTIONS

There are a few ways of creating an analogue output with a Pico. By far the most common is to use pulse-width modulation (or PWM). This flicks a digital output on and off very quickly so that it averages out to some point in the middle. By controlling how long it spends flicking up and down, you can control where in the middle it averages out. This works well →

Below
You can make the circuit (semi-) permanent by gluing the components upside down and soldering to the legs. This technique is known as 'dead bug' because the pins in the air look like a dead bug's legs



Above
We're experimenting with 3D-printed cases – in this iteration, the text is too small to print cleanly. Once we have a version that we're starting to be happy with, we'll add them to the repository

ASSEMBLY

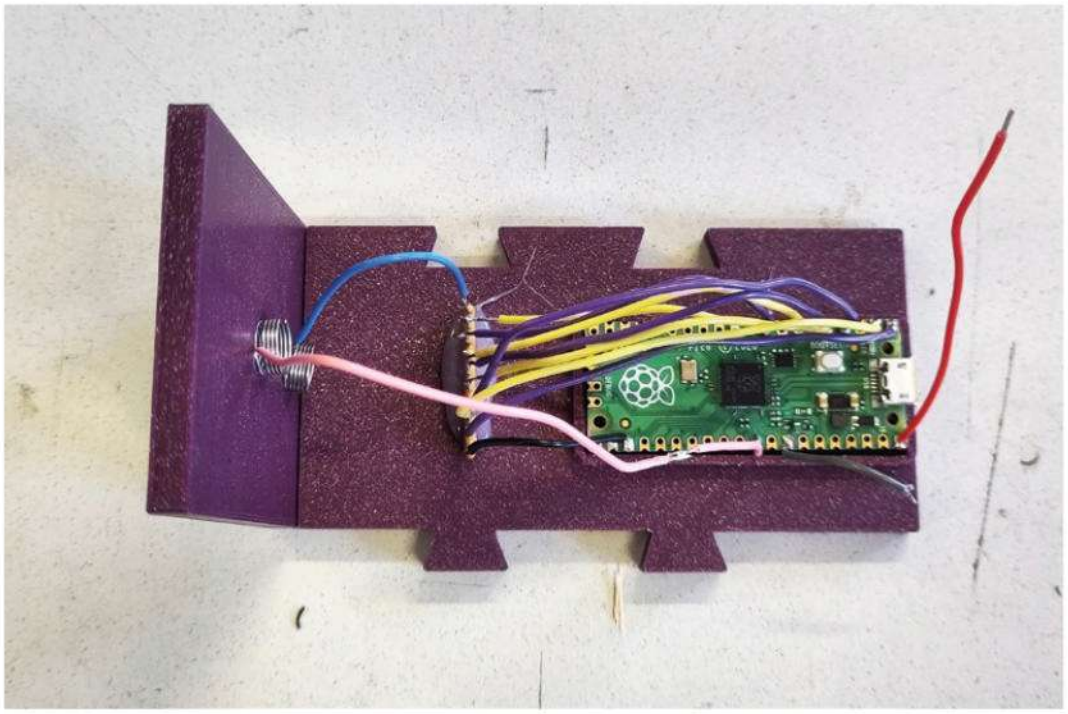
When a project is made up of lots of little parts like this, the cost of any one of those little parts can have a big impact on the final price. One question we've wrestled with over this is how to hold the components together. There aren't many of them, and we could easily do it on stripboard or protoboard. We could even design some PCBs to hold everything together. However, either of these options would add a sizeable chunk onto the final cost.

We'll look at the cost of these options in a future article, and once you've got a setup that you like, it could well be worth doing. However, in the interests of making a cheap, hackable, accessible musical instrument, we're going to go with 'dead bug' style circuits. With this, the components are glued upside down and connections are soldered wires. It's a bit messy, but fairly secure and easy to modify.

for dimming LEDs, and can even create quality audio. However, it doesn't work for our purposes because you can't reliably read it with another microcontroller.

The simplest way of getting true analogue out from a microcontroller is to use an R-2R resistor ladder. This is a combination of resistors (some of which have twice the resistance of the others, hence the name). It's cheap and easy to use, but any inconsistencies in the resistors add up. As such, it's hard to get more than seven or eight bits of resolution. This is fine for some purposes – you can create a reasonably accurate waveform from these. However, if you're using a voltage to set the pitch, this lacks the resolution needed, particularly for pitch bends and finer control.

The most reliable way to set a voltage is with a dedicated digital-to-analogue converter (DAC). These allow us to specify the voltage we want. However, they are the most expensive option. They are also slightly harder to control.



Left
Multiple modules will connect together like a jigsaw, letting you create a synth out of an arbitrarily large number of modules

> **You can either make this resistor ladder out of** resistors, or buy one as a dedicated bit of hardware

We'll look at dedicated DACs in the future, but for this article, we'll use a simple resistor ladder.

THE FIRST MODULE

Music all starts with oscillations. Sound in the air is a compression wave, and this is encoded electronically as a voltage varying over time.

Modular synths control sound using control voltage. This is an analogue signal that defines the sound, with one volt range corresponding to one octave range of audio.

The link between control voltage and sound is a voltage-controlled oscillator (VCO). This reads in a voltage and outputs a sound-wave at a particular frequency.

Our first module will then be a VCO. The hardware for this is a Pico (a Pico W should also work, though we don't use the wireless hardware) and an R-2R resistor ladder. You can either make this resistor ladder out of resistors, or buy one as a dedicated bit of hardware (we used a 4610X-R2R-103LF from Bourns). The operation is the same – if you already have 26 resistors of the same value, it's probably worth using them, otherwise, it's easier and cheaper to use dedicated hardware. That said, we'd caution against buying too much hardware now as you might find it easier and cheaper to wait until we've got the bits for a playable synth.

To wire it up, you just have to connect ground, then pins 0 to 7 from Pico to the R-2R ladder, and the output from the ladder is the module's output.

The output of this is an audio signal in the 0–3.3 V range. It can't source enough current to power more than a small speaker, yet it's too much voltage for line-level inputs. For the purposes of testing it out, you can clip it onto a pair of headphones. We'll look at creating a more standard output in the future.

In this case, it can just be connected to a jumper wire and crocodile clips to the top ring of your headphone's jack (and the bottom of the headphone jack can go to ground).

The input for the VCO is Pico pin 26. For now, you can attach this to ground.

SOFTWARE SETUP

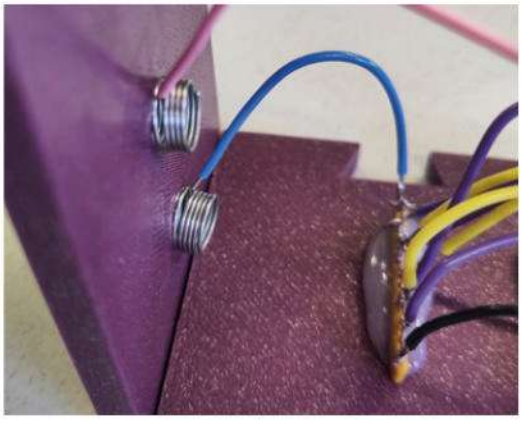
We want our modular synth to be hackable, and we also want it to perform as well as it possibly can for the given hardware. After going through the various options (including CircuitPython, MicroPython, and Arduino), we've decided that the best option is to use the Pico SDK and program it directly in C. This is a little more complex to get started in, but it does give us a lot of flexibility in how we use the hardware – flexibility that we'll be taking advantage of in future modules. →

If you don't want to get your hands dirty with code, we'll also provide UF2 files that you can upload directly to Pico.

All the assets for this project can be found at **github.com/benevpi/PicoModular**. In the binaries folder, you'll find a UF2 file for PicoVCO_R2R. Hold down the Boot button on Pico and then plug it in via USB (unplug first if necessary), and you should see a USB drive called RP2 appear. Drag and drop the UF2 file onto this drive, and it will upload the project onto your Pico and (if the hardware is all attached) you should hear a note on your headphones.

The code that powers our VCO is:

```
#include <stdio.h>
#include <math.h>

#include "pico/stdlib.h"
#include "pico/multicore.h"
#include "hardware/adc.h"
#include "eightbit_r2r_dac_simple.pio.h"
#include "waves.h"


#define WAVESIZE 2000
#define PEAK 255
#define START_PIN 0
#define LOWEST_FREQ 100
#define CYCLES_PER_PIO_LOOP 3
#define CPUFREQ 130000000

int triangle_wave[WAVESIZE];
PIO pio;
uint sm;



float calc_clkdiv(float frequency, int length) {
        return (CPUFREQ / (frequency*CYCLES_PER_
PIO_LOOP*length));

}

void core1_loop() {
        while(true) {
                for(int i=0; i<WAVESIZE;i++){
                        pio_sm_put_
blocking(pio, sm, triangle_wave[i]);
                }
        }
}

void main () {
        stdio_init_all();

        adc_init();
        adc_gpio_init(26);
        adc_select_input(0);
        const float conversion_factor = 3.3f / (1
<< 12);

        generate_triange(triangle_wave, WAVESIZE,
PEAK);

        pio = pio0;
        uint offset = pio_add_program(pio,
&eightbit_r2r_dac_simple_program);
        sm = pio_claim_unused_sm(pio, true);

        float init_clkdiv = calc_clkdiv(440,
WAVESIZE);

        eightbit_r2r_dac_simple_init(pio, sm,
offset, START_PIN, init_clkdiv);

        multicore_launch_core1(core1_loop);

        float divider;
        float freq;
        while(true) {
                uint16_t result = adc_read();
                freq = LOWEST_FREQ * pow(2,
result*conversion_factor);
                divider = calc_clkdiv(freq,
WAVESIZE);
                pio_sm_set_clkdiv(pio, sm,
divider);

        }

}
```



Right
We're still experimenting with the best way of connecting inputs and outputs together. This spring arrangement will be familiar to anyone who learned electronics in the early 1990s

As you can see, this uses a simple PIO program to continually put data on the GPIO pins. We could do this directly from C, but PIO has very accurate timing, so it becomes simple to adjust the pitch of the note by adjusting the frequency of the PIO state machine (which you can see happening in the main loop).

The PIO program is:

```
loop:
        out pins, 8
        pull ; could do autopull, but this feels
less error prone if changing the size
        jmp loop
```

This could be more concise, but it's helpful to have more instructions because there's a limit to how much we can slow down the state machine, and having three instructions per loop gives us a better range of frequencies.

We're also creating a library of different waveforms. At the moment, this just has a single function, `generate_triangle()`, which unsurprisingly generates a triangle wave. In the future, we're going to expand this to more waves. The program is just putting values from an array onto pins, so there are no limits on the waveforms that it can produce.

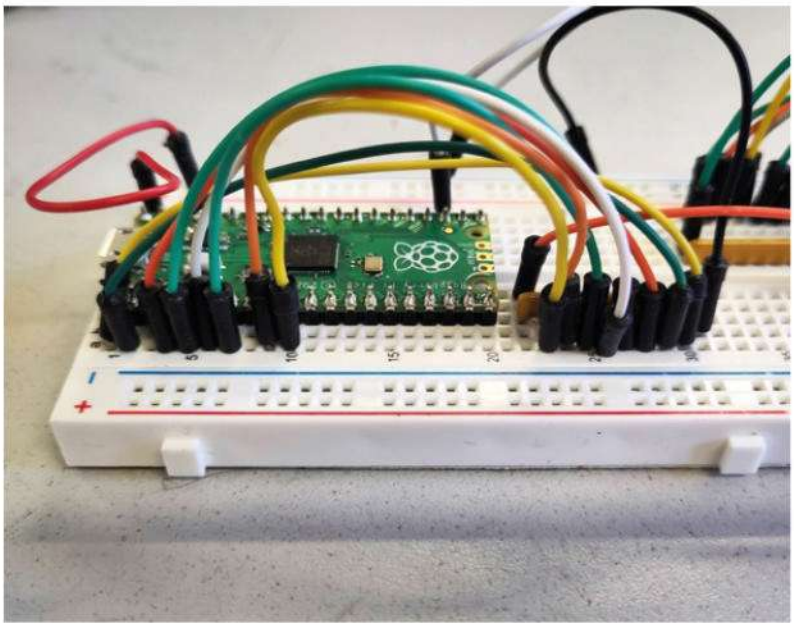
> We've only built one type of module, but we can already **start to look at how to generate sound with it**

This just outputs a triangle wave, but there's nothing in the hardware that limits the wave shapes that it can output, and we'll investigate more in the future. In this series, we're going to focus on getting a playable instrument, then look to add complexities and intricacies later. Since it's designed to be hackable, we shouldn't have a problem with that.

We've only built one type of module, but we can already start to look at how to generate sound with it. We can use one of these as a low-frequency oscillator (LFO). LFOs output a waveform that's below audio range, and are used to control things rather than play sound directly. In this case, it means we can use the output of the LFO as the input to the VCO to create a warbling sound.

We've also created a UF2 for a voltage-controlled LFO. This is almost identical to the VCO code, but with a couple of tweaks to make it output a much lower-frequency wave. If you take a second Pico and wire it up in exactly the same way as the VCO Pico, but connect the output from the R-2R ladder to the analogue input of the VCO Pico, then you'll hear the frequency of the VCO change up and down with the wave from the LFO. It's not the most complex bit of sound synthesis, but hopefully it will give a little glimpse of what our Pico modular system will be able to do.

We honestly don't know how far this project will go. It's an experiment in cost engineering and pushing the analogue capabilities of fundamentally digital devices to their limits. We're hoping to end with something that's enjoyable to play, and helps you learn about music and the hardware behind it.



Above
To get an analogue output, connect the low end of the R-2R ladder to ground, then GPIO 0 to 7 in order, and then the output is on the final end

WHAT DO YOU WANT TO SEE?

We have a plan for how we want to build our synth, but all plans are flexible. We'd love to hear what you'd like to see in a Pico modular synth. Are there specific modules you'd like to build? Specific bits of hardware you'd like to interface with?

Let us know. We can't promise to accommodate everything. We can't even promise that it's possible, but we'd love to hear about it. Email your ideas to **ben.everard@raspberrypi.com** and we will take them into consideration.

reBartender V0.1

Get yourself a Raspberry Pi-powered drinks dispenser with this cool setup by Seeed Studio and their reTerminal

MAKER

Peter Pan

An application engineer at Seeed Studio, he's also a maker in his spare time, so he fits right in with the maker atmosphere of Seeed Studio.

seeedstudio.com

You'll Need

- reTerminal DM **magpi.cc/reterminaldm**
- 150 mm of 20 mm × 20 mm aluminium frame
- 4 × right-angle brackets to fit
- Nuts and bolts
- 3D-printed parts
- 4 × 12 V DC peristaltic pumps
- 12 V power supply
- Various jars, pipes, metal straws, and drinks to fit
- STL and code files: **magpi.cc/rebtgit**

During the celebration of the Dragon Boat Festival at Seeed Studio, a booth was set up to serve drinks with an automated cocktail machine and named reBartender V0.1.

The prefix 're-' is commonly used to mean 'again', and Seeed Studio created a product line called reThings to evoke 'redefine', and includes products like reTerminal (which we reviewed in *The MagPi* 122 (**magpi.cc/122**) and the latest reTerminal DM (Device Master). They're both powered by Raspberry Pi Compute Module 4 and have a lot of industrial interfaces – perfect for attaching to pumps, pipes, and more to help serve some cocktails. Let's get mixing.

01 Cut the aluminium

The aluminium frame needs to be cut into 2 × 15 mm lengths for feet, 2 × 25 mm for the legs, and a 70 mm piece for holding reTerminal DM with DIN rail. The peristaltic pumps are just held on by cable ties.

See **Figure 1** for the basic build – the feet are laid lengthways on the desk, and right-angle brackets are used to attach the legs, which are orientated upright. The long piece bridges the two, again attached by the brackets, and that's where we'll attach reTerminal later. Be careful using power tools for this – not only are they dangerous, but cut metal can be very sharp.

Figure 1: The basic build of the frame, once done

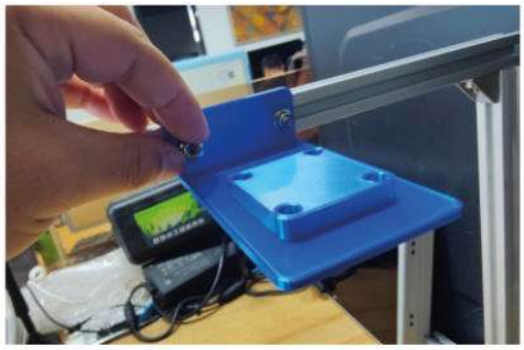


Figure 2: Affix the straw holder onto the aluminium frame

02 Attach pipe holder

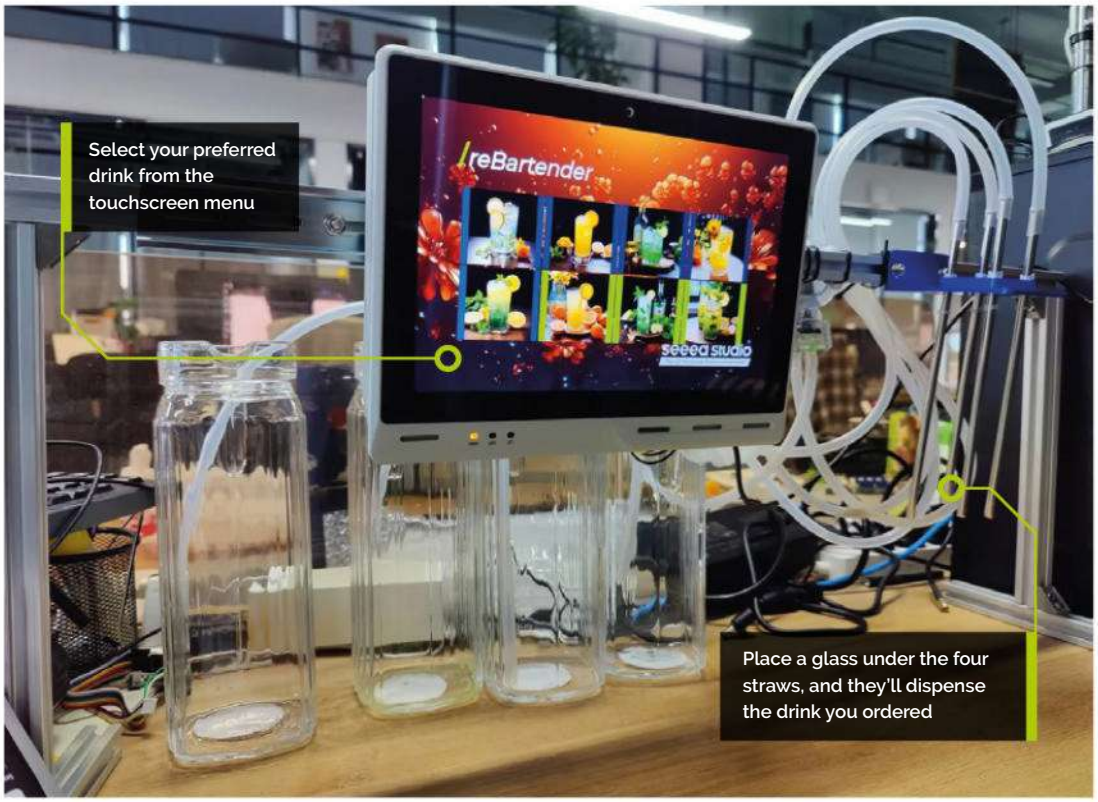
For the pipes attached to the pumps, the input end goes to a jug which will be filled with the various cocktail ingredients, and the output end of each silicon pipe goes into a metal straw. To hold the straws in a fixed position so that patrons can easily fill their glasses, a pipe holder is 3D-printed and attached to the frame.

Affix the short end to the frame with nuts and bolts (**Figure 2**), and then insert the straws into the pipes and through the holes on the 3D print.

> "Affix the short end to the frame with nuts and bolts"

03 Wiring it up

Figure 3 shows the wiring diagram for the setup. The DM Terminal is on the rear of reTerminal DM and has four opto-isolated DO (digital output) interfaces that can directly control a load of 500 mA. In this case, we can connected four peristaltic pumps with wiring to the reTerminal DM 20-pin terminal, where each individual pump can be controlled by the DO pins. The pumps used are driven by a 12 V DC motor, and the maximum current they draw is around 200mA.



Warning! Power tool safety

This project uses power tools. Be careful and wear safety equipment.

magpi.cc/powertools

04 Preparing to code

We're using Node-RED to create a 'proper cocktail mixing flow control algorithm', which is definitely real and important computer science, along with a nice UI for people to interact with.

The 20-pin terminal that we're using does correspond to specific GPIO pins on Raspberry Pi. You can find out more details on the entire terminal here: **magpi.cc/rtdmpins**. For our uses, the DO pins are labelled as such:

DO1 – GPIO21
DO2 – GPIO25
DO3 – GPIO26
DO4 – GPIO6

The fifth pin that we've plugged into is a ground pin for the DO pins.

05 Setting a pin in Node-RED

From the Raspberry Pi nodes, select rpi-gpio and add it to the flow. Double-click it to change the settings: pin needs to be 18 – GPIO24, BCM GPIO should be 24, type is Digital output, initialise pin state should be ticked, and initial level of pin should be low (0). You can double-check the settings next to **Figure 4** and, once you're happy, click Done. Repeat this for GPIO25, 26, and 06.

06 Flow control

Let's create the main algorithm. We can use the 'trigger node' to send messages with a short delay – the core algorithm behind the ingredient volume control. We can use 'change node' to send the stop signal after the delay period when chaining together more pumps.

In the settings for the trigger node, select 'true' for Send. Select 'wait for' for then and set the waiting period. This version uses 1000 milliseconds as it's about a 400 millisecond delay with the pump used to fill up a single shot glass. Set 'then send' to

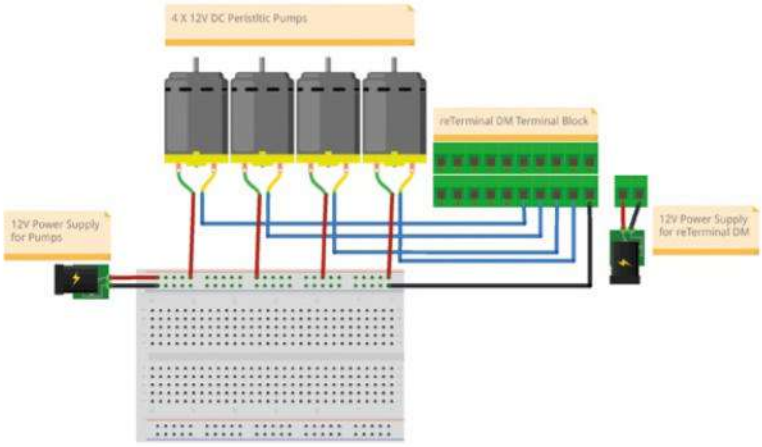


Figure 3: The wiring is fairly simple. Make sure the ground for the pump power is connected to reTerminal

Top Tip

Cocktail trial and error

Got a favourite cocktail? It usually has a very specific mixture of spirits and mixers, so getting the time delay correct for the right amounts will take some tweaking.

'true' for triggering next pump(/ingredient) within the same recipe. Tick 'send second message to separate output' and then hit Done.

For the change node, you'll need to open the properties and choose 'Set', 'msg.payload', and then to the value is 'false'. This allows one ingredient to be dispensed at a time.

07 Button nodes

Add a button node and open the properties. Set the group to '[HOME] Default', set the size to 6 × 5, and scroll down to 'When clicked, send' – set the payload to 1 which will be the number the button sends when pressed. Click Done.

We need to let Node-RED know we're using the UI to control stuff. Add a ui control node, open the settings and select JSON (or {}) as Send and then enter {"tab":"Overlay"}. Choose 'wait for' from the then menu, and enter 10000 milliseconds.

'Then send' should be JSON/{} again and this time enter {"tab":"HOME"}. You can check what you've done against **Figure 5**, and then click Done. Finally, add a ui control node, and set its output to 'Change tab or group events only'.

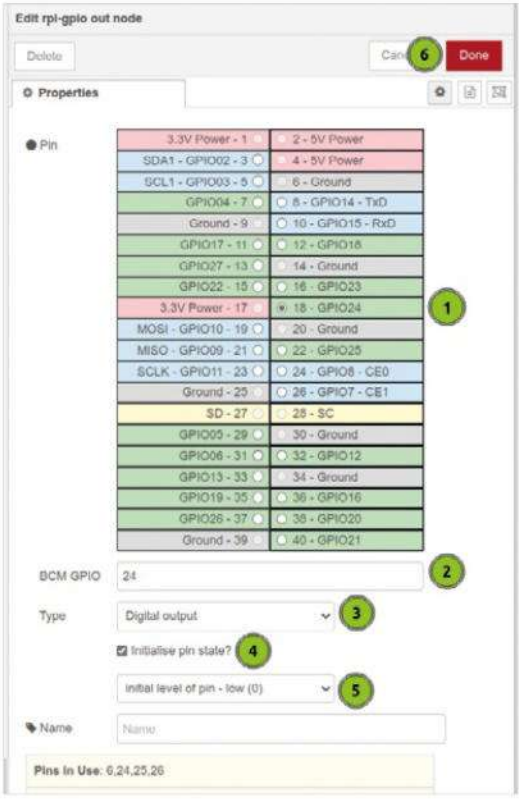


Figure 4: How to set up the GPIO pins in Node-RED

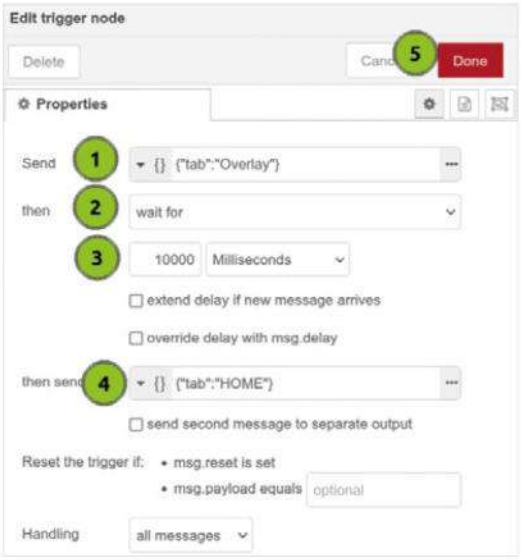


Figure 5: The settings for the trigger node

08 Record button presses

Add a csv node to the flow and open up the settings. For this, we want the Columns to be button, button_num, and comma as the Separator. In the input section, we want 'parse numerical values' ticked. Click Done when you're happy.

Now we need to write to the file. Add a write file node and set the file name to a path called test.csv. The Action should be append to file and then 'Add newline (\n) to each payload?' should be ticked. You want to now link the output of the csv node to the input of the write file node.

09 Main flow

With the nodes set up, we can start creating recipes by linking up the nodes! Check **Figure 6** for an example recipe, in this case a vodka and soda. With several recipes added to the flow, you'll get a flow like **Figure 7**. The recipes use sparkling water (pump one), vodka (pump two), orange juice (pump three), and mint syrup (pump four). Here are the original recipes:

Recipe 1 (Vodka and Soda): Sparkling water × 1000ms, vodka × 400ms
Recipe 2 (Screwdriver): Vodka × 400ms, orange juice × 800ms
Recipe 3 (Vodka Mojito): Sparkling water × 800ms, vodka × 200ms, mint syrup × 400ms
Recipe 4 (Orange Crush): Sparkling water × 800ms, vodka × 200ms, orange juice × 400ms
Recipe 5 (Vodka Collins): Vodka × 200ms, orange juice × 800ms, mint syrup × 400ms
Recipe 6: Sparkling water × 1000ms, orange juice × 400ms

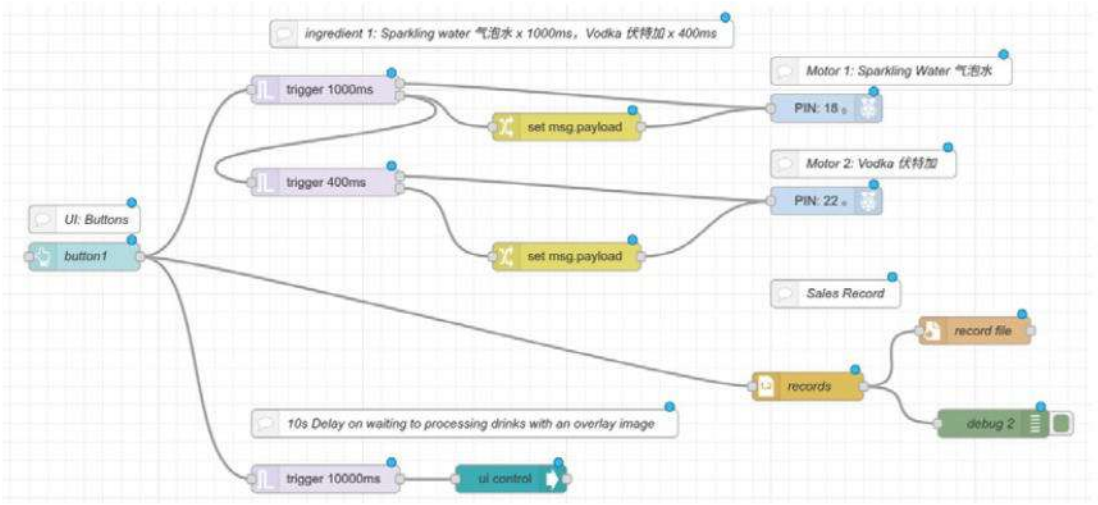


Figure 6: An example recipe flow

> We can set the buttons and other parts of the interface to use images

Recipe 7: Sparkling water × 1000ms, mint syrup × 400ms
Recipe 8: Sparkling water × 1000ms, orange juice 200ms, mint syrup × 200ms

10 UI folders

To make a nice-looking UI, we can set the buttons and other parts of the interface to use images. This is quite simple to do – first, open the terminal on Raspberry Pi and get to the Node-RED settings folder with:

```
cd ~/.node-red
```

Then open the **settings.js** file with:

```
nano settings.js
```

There will be the line `//httpStatic: '/home/pi/node-red-static/', //single static source`. Remove the // at the start to uncomment the line and save the file. Finally, use `cd ~` to head to the home folder and use:

```
mkdir node-red-static
```

Put the images for the drink buttons and background you'd like to use in the folder, and give them a simple name – e.g. button1, button2, background, etc.

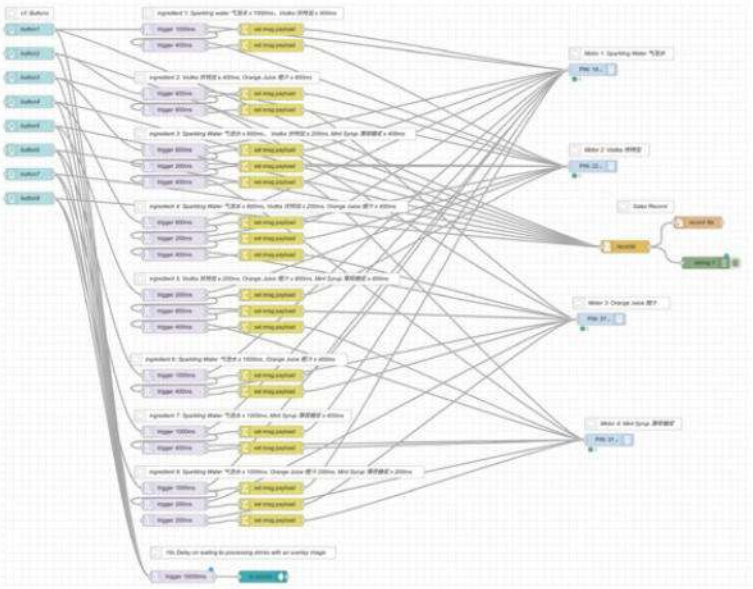
11 UI code

With that all done, you can add a template node to the flow and add CSS styling, linking the node id of the buttons to the images, and including the background. You can find the code used in this project at **magpi.cc/rebtgit**, as well as some example images if you're having trouble finding some yourself.

12 Automated sip

You're done! Fill up the jugs with the desired ingredients, and get ready for your automated bartender to serve you some drinks. Remember, drink responsibly – that way you can keep the containers filled up without any spills or accidents.

Figure 7: Full flow for eight buttons using four ingredients



RASPBERRY PI 5

Priority Boarding

We've reserved Raspberry Pi 5 boards
for HackSpace magazine subscribers

GET YOUR RASPBERRY PI 5 NOW!

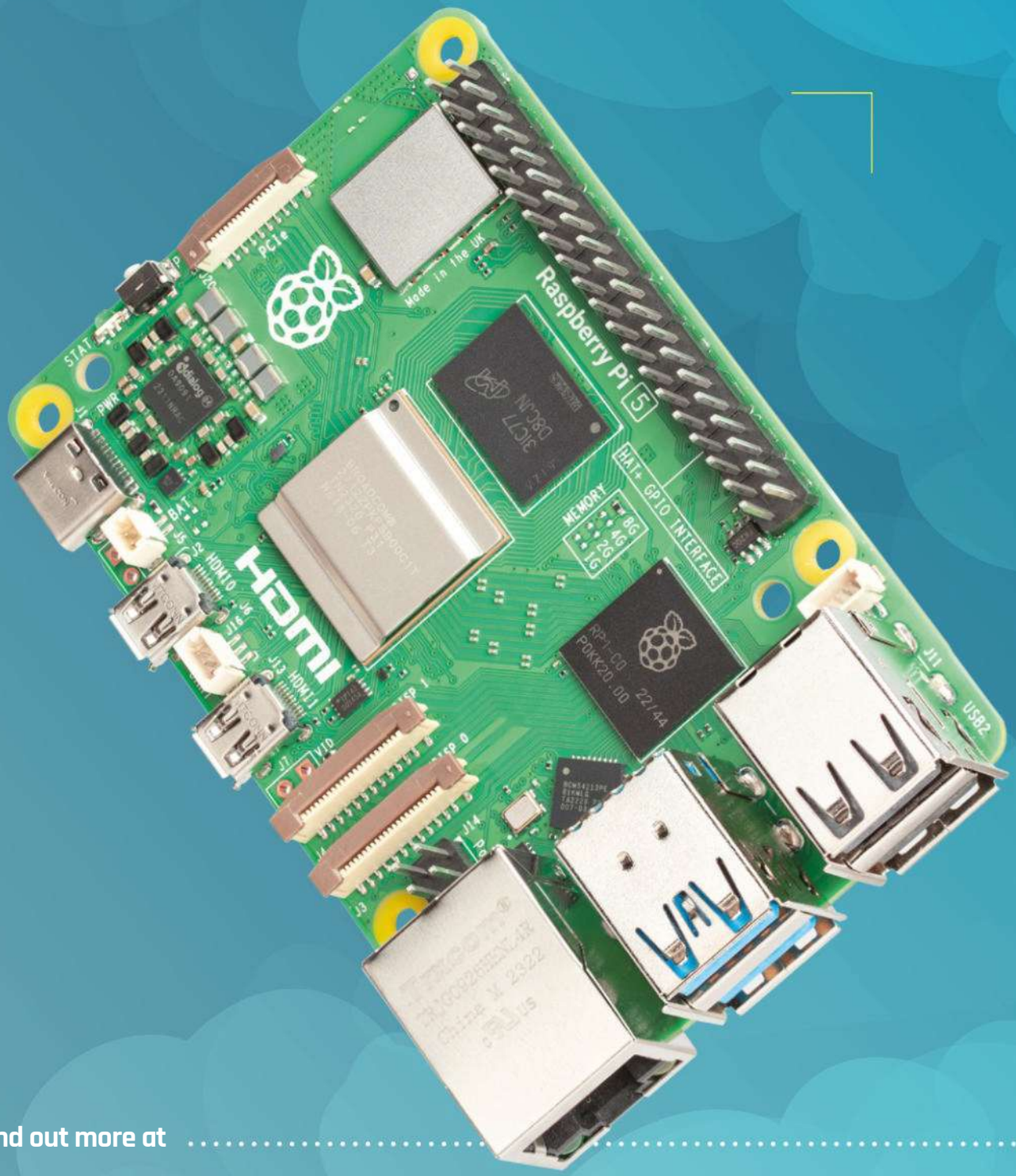
Want to get Raspberry Pi 5 sent to you right now, without waiting for stock? With Priority Boarding, you can order your Raspberry Pi 5 (4GB or 8GB) and it'll be sent out right away. Raspberry Pi has set aside thousands of Raspberry Pi 5 computers for The MagPi and HackSpace magazine print subscribers to buy. Enough to guarantee every subscriber can get one without going out of stock.

How it works!

If you take out a print subscription to HackSpace magazine or The MagPi, we'll send you a unique code in the next few days. Along with the code, you'll get details of how to use it through our partners (Pi Shop in USA and Canada and the Pi Hut everywhere else).

Get started

New subscribers to HackSpace mag will get a code when they sign up. So don't delay; take out a subscription today (**hsmag.cc/subscribe**). You'll be able to buy your Raspberry Pi 5 first and get a magazine packed full of incredible maker tutorials, projects, and reviews.



Find out more at

hsmag.cc/prioritybording

TUTORIAL

Creating with eco resin

A safer and more environmentally friendly alternative to epoxy resin, eco resin can produce some stunning results

Nicola King

@holtonhandmade

Nicola King is a freelance writer and sub-editor. Madly making things for a Christmas fair stall – her sewing machine, knitting needles, and pliers are working at speed.

Eco resin is arguably the new kid on the block in terms of resins, with long-established epoxy resins having been popular for some time in the world of crafting. In fact, we've even seen makers embedding LED lights in the substance (**hsmag.cc/LEDsInResin**), to great effect.

The difference with eco resin, however, is that it's just a little kinder to both the user and the environment while still being strong and durable, and it is gaining traction as a go-to compound for makers of all abilities. It also gives a very different finished effect from epoxy resin, with a far more ceramic/stoneware effect, and you can incorporate glossy, matt, or marbled finishes. So, if you want to bypass using more toxic substances in your making, read on for some artful and carefully cured inspiration as, in this tutorial, we are going to make some eco resin coasters, ever-useful little mats for your beverages to stand on.

Right
Eco resin is growing in popularity, and the range of items you can create is huge. Entice out the artist in you with various decorative techniques

STEP 1 MEASURE METICULOUSLY & MIX

With your work surface prepared and your mould chosen, if you want to line the bottom of the mould with some form of decoration, e.g. silver leaf, beads, shells and so on, now is the time to do that before you start mixing your resin. It's really important, when casting eco resin items, that you use scales to accurately measure out the amounts of resin casting compound and water required. Otherwise, your mixture won't cure properly and will be an expensive waste of money, mixture, and time. Clear instructions on how much you need to weigh out per 100g are given on the pack – ours required 35g of water for every 100g of resin compound. In terms of how you work out exactly how much resin you need for a specific mould, weigh your mould empty, fill it with water, and then weigh it again. The difference between the two weights is the volume, and you can use this to figure out how much water and compound is required.

Once you have your powder and water together in a mixing cup, ensure you mix until it is free of lumps and has a creamy, batter-like consistency. You have 10–15 minutes to work the mixture, so have some colours ready to add.

Left
Some moulds filled and ready to cure. Make sure you have some spare moulds on hand as, if you have mixture left over after filling your main moulds, you then won't waste anything

QUICK TIP

This is a craft that can get a little messy – ensure that your work surface is protected, and wear gloves to minimise clean-up time later.

YOU'LL NEED

- **Eco resin compound (water active),** e.g. hsmag.cc/CastingResin
- **Silicone mould(s),** e.g. hsmag.cc/ResinMoulds
- **Digital scales**
- **A silicone or plastic mixing container/cup** (ideally with a spout for pouring)
- **A stirrer/mixing stick and spoon**
- **Colour/decoration,** e.g. acrylic paint, mica powder, pigments, waxes etc.
- **Water**
- **Gloves**
- **Old newspaper** (to protect table surface)
- **Sandpaper** (for wet sanding) and beeswax/varnish (for finishing)

> It's just a little kinder to both the user and the **environment while still being strong and durable**

STEP 2 COLOUR KALEIDOSCOPE

It's over to you regarding the colour that you want to add at this stage. We've seen jet black makes that look incredibly effective with some gold leaf inserted into the design. We have used some inexpensive acrylic paint and mixed it in, with a light mixing resulting in some pleasing marbled effects. The more pigment that is added, obviously the more intense the colour that results, but add the colour slowly until you achieve the result that you are aiming for, as it's far easier to add more than to tame down the colour if you go overboard. Also, you need to be aware that adding any colour will affect the setting time of the piece, so don't add too much or you'll find it will take way longer to set or not set at all. We certainly learnt a few lessons in this during our first few mixes, and you definitely discover a great deal from a few practice rounds.

CREATING EYE-CATCHING EFFECTS

There are many options when it comes to creating some fantastic colourful effects in your eco resin projects, as it's such a versatile medium. Here are just a few ideas:

- Any water-based pigment, such as acrylic inks or paints will work well. These have a smooth consistency and mix really easily with the resin compound. If you want a marbled effect, add several drops of paint and gently swirl it a few times with your stirrer, but don't fully mix the paint in.
- Mica powders, or some glitter, are also very effective, and you really don't need to use very much to create something beautiful. We used some metallic pigment colours, and loved the effect: **hsmag.cc/MetallicColour**.
- Embed some solid shapes into the mix, such as shells, sand, small mosaic tiles, tiny beads, crystals, jewellery charms, or natural gemstone chips. They look particularly effective in the rims of pots or coasters, and give it a great texture as well as colour, making your project completely unique. How about placing some dried flowers in the base of a coaster mould, then pouring the eco resin over the top?
- Silver or gold leaf can also be used in the base of a mould, with the resin poured over the top, and the effect when your project is de-moulded is stunning.

TUTORIAL

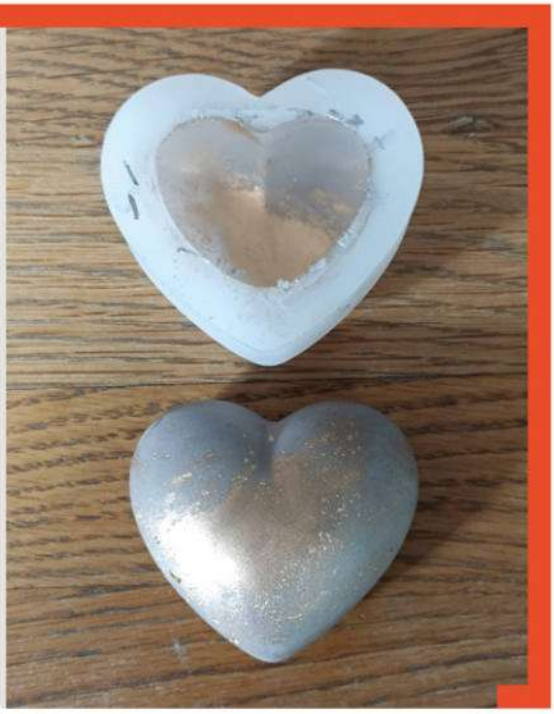
ECO RESIN VS EPOXY RESIN

We have used eco resin in this tutorial for a number of reasons, the main one being that it is arguably a much safer product to work with than epoxy resin. While epoxy resin is undoubtedly popular to craft with, and creates great-looking end products, there are a couple of key factors that you need to bear in mind when you are handling it, including:

- Chemical makeup: Despite the fact that formulations in epoxy resin have come a long way over the last ten years or so, this resin is essentially a petroleum, chemical-based product, and the chemicals can affect your health if they come into contact with your skin, or if you breathe them in. They give off vapours which can cause allergic reactions (especially if you suffer from asthma) and can ignite if not handled correctly. This resin is more toxic in its liquid form than in its solid form. When working with epoxy resin, you should wear gloves, safety glasses, protective clothing, and a respirator mask to safeguard yourself from the vapours. Ideally, you should work in a very well-ventilated space, with open windows to help disperse the epoxy vapours.

- Synthetic epoxy resins are not particularly environmentally friendly, and won't biodegrade as easily as more eco-friendly resins.

Eco resins, in contrast, are water-based, a type of bio-based thermosetting plastic, often made from plant-derived oils and natural fibres. They are non-toxic and have a lower environmental impact and reduced carbon footprint, and some are biodegradable and compostable. Eco resin behaves very much like concrete, with similar strength properties, but you arguably have much less to worry about in terms of how it will affect both your health and the health of the planet.



QUICK TIP

To reduce bubbles, stir the mix slowly and definitely avoid aggressive mixing.

Right
We left the residue mix to harden in the plastic cup and then flaked it off into terrazzo chips

STEP 3 ...AND POUR

When you are happy with the mixture, you can pour it into the mould(s). Go slow and steady and try not to waste anything, so any overflow should go into another mould. Alternatively, you can pour the remainder onto a flat surface, and onto something like a plastic sheet, spread it thinly with an old lollipop stick, for example, let it dry, and later break it up into little 'chiplets' of colour that can be used in future eco resin projects. These are called terrazzo chips.

You'll find that air bubbles naturally rise to the surface of the resin mix. Gently give the mould a little tap or two, and the air bubbles will disperse, or you can pop them with the end of a stirrer. When you are happy, leave the resin to set. As an example, our instructions told us to leave it for 60 to 90 minutes to harden. You can then de-mould. However, allow to dry for a further 24 hours to reach optimum cured strength before you do anything else to it.

Left
There are various sealing options, from natural finishes such as beeswax or coconut oil, to water-based acrylic sealers. If your piece will be outside, use a stronger sealer

QUICK TIP

Clean your silicone moulds before the next use. Wash in a bucket of soapy water and make sure that any residue is thrown in the bin, not washed down your sink, or your pipes will get clogged!

STEP 4 FINISHING TOUCHES

Next, we need to sand the piece down to get rid of any superfluous bits of resin around the edges, but also to bring through the true colour of the piece, especially if terrazzo flakes have been incorporated, as sanding makes them more prominent. Note that wet sanding is recommended (so you'll need to use a sandpaper that is designated for wet sanding), as it removes the dust from your work as you go, and you might want to wear a mask when sanding to avoid breathing in fine particles. Follow sanding with a coat of some kind of sealant, and this is particularly important if you are making plant pots for example. This coating will add strength, waterproofing, and generally prolong the life of your make. You can buy clay sealing agents or acrylic varnish, but we used some beeswax on a couple of pieces, which worked well.

So, we hope you give eco resin a try, as this author has found that not having to worry about toxic ingredients really enhances the making experience, making it a suitable craft for all ages.

WHAT CAN I MAKE?

The versatility of eco resin means there are plenty of things you can whip up. The ceramic or concrete-like appearance lends itself to contemporary:

- **Decorative homeware** – think ornaments, candle-holders, doorstops, plant pots, trays, trivets, egg cups, vases, tiles etc.
- **Jewellery** – mould some earrings, add some colour and findings, and you have a beautiful gift. Make charms for necklaces, bracelets too, key rings, or fashion some hair attire.
- **Make some functional items** like buttons using a mould, a bookmark or a paperweight, or some knobs for drawers.

There are a plethora of books on the subject, and we found *Eco-resin Crafts* by Hazel Oliver to be really useful. YouTube is also worth a search for relevant hints and tips.

TUTORIAL

Turn an old phone into a robotic personal assistant

Bring an old phone into the 21st century by adding a Raspberry Pi to turn it into a networked assistant

Rob Miles

Rob Miles has been playing with hardware and software since almost before there was hardware and software. You can find out more about his so-called life at **robmiles.com**.

Rotary dial phones are fashionable again. You can even buy brand new 'old' ones based on the original design shown in **Figure 1** below. These have dials, but not the weight or the authentic bell sound. The author was completely unaware of this trend when he picked up his red phone. The idea was to retain the external appearance and behaviours but bring the device up to date with all-new internals and some fun behaviours. It seemed to him that there should be room inside for a reasonable amount of computing power, and he was keen to hear the old telephone bell sound again. He wanted to create a web-controlled device that could be used to receive messages and alerts. The telephone that was built contains a Raspberry Pi Zero 2 running JavaScript code inside the node environment, using Express to host a telephone website. You can find all the code, 3D files for the mounting plate, and a setup sequence for a Raspberry Pi in the GitHub repository for the project here: **hsmag.cc/DialTelephone**.

Figure 1
The phone still looks good nearly 50 years after it was made

DELVING INTO HISTORY

The first task was to open the telephone and check the amount of space available for the new innards. **Figure 2** shows the printed circuit board (PCB) inside the phone. This design was one of the first times a PCB had been used in a UK telephone.

The author would have liked to have kept the internal components in place so that the phone could be returned to its original state if required. Unfortunately, this turned out to be impossible.

WIPING THE SLATE CLEAN

The circuit board was cleared of components, and a 3D-printed holder for the Raspberry Pi Zero with prototyping board was inserted into the space, as shown in **Figure 3**, overleaf. The phone will use a 12-volt power supply, and it was found that a power supply socket fits into the cord holder for the exchange connection with an appropriate washer.

Figure 4, overleaf, shows the circuit for the telephone. The two devices in the centre are two 'buck converters'. The one on the left converts the 12-volt power input into 35 volts to power the bell. The second converts 12 volts into 5 volts to power the Raspberry Pi. The handset switch is connected to the handset cradle and indicates whether the handset is on the phone. The dial pulse and dial active switches are in the telephone dial, of which more later.

RING THE BELL

The bell in the telephone was originally driven by two coils powered by a 75-volt alternating current signal with a frequency of around 18Hz. The coils move a bell-clapper left and right between two metal bells tuned to different musical notes. The author was very keen to retain the distinctive ring, but less keen on getting 75 volts up his armpits when assembling the phone. So, rather than using 75 volts AC, he opted to

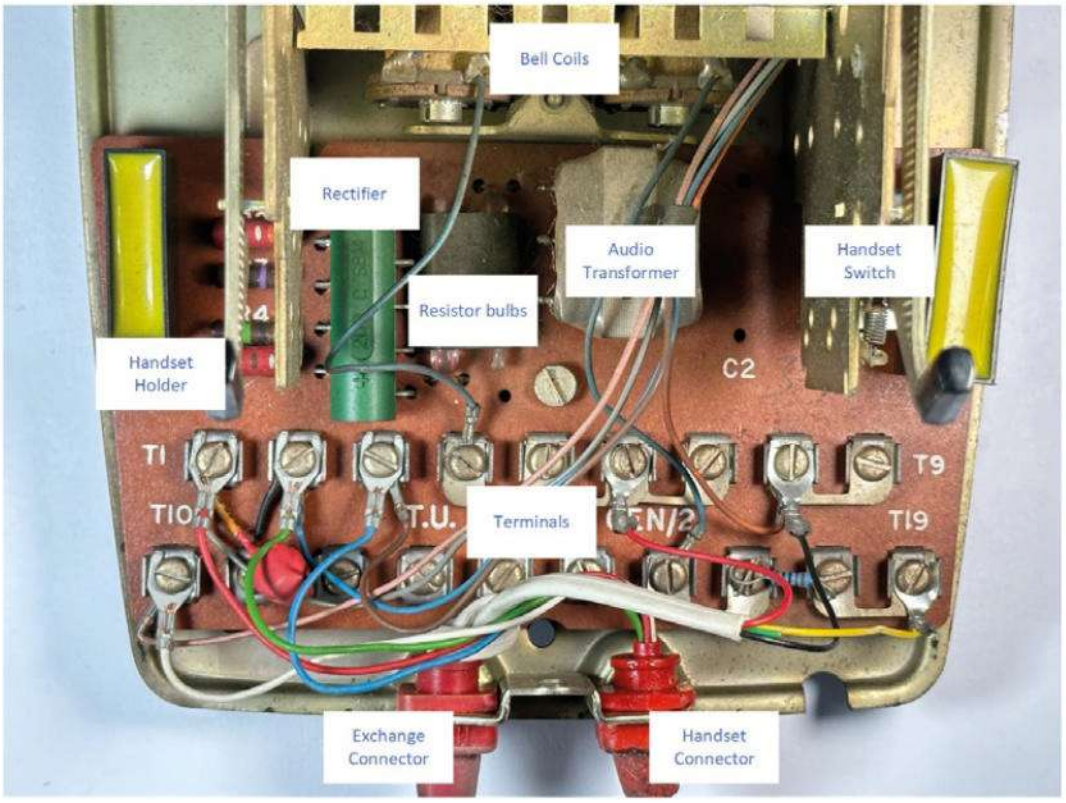


Figure 2
The circuit sends audio signals through the resistor bulbs if it detects the phone has a connection which is close to the telephone exchange. This reduces the sound volume and makes the bulbs light up in time with your speech

use a much less tingly 35-volt supply, using software to drive each coil in turn. Two MOSFET controllers were used, one for each bell. These are connected to general-purpose input/output (GPIO) pins on the Raspberry Pi which are controlled by JavaScript running the phone.

```
async ding() {
    this.bell1GPIO.on();
    await this.delay(25);
    this.bell1GPIO.off()
    this.bell2GPIO.on();
    await this.delay(25);
    this.bell2GPIO.off();
    return;
}
```

A single 'ding' is produced when the handset is lifted or replaced, just like the old phones do. The JavaScript above makes the bell go 'ding' when the `ding` method is called. The code moves the clapper towards each bell in turn. The delay values of 25 milliseconds between the movements of the clapper were determined by trial and error.

```
async repeatRing(length) {
    for (let i = 0; i < length; i++) {
        await this.ding();
        if (!this.ringing) {
            return;
        }
    }
}
```

The `repeatRing` method produces a longer ringing sound by repeatedly calling the `ding` method the requested number of times. It checks the `ringing`

TELEPHONE ETIQUETTE

If you've only ever seen mobile phones up to now, you might be wondering how a dial phone is used. If the handset is on the phone, as seen in **Figure 1**, the phone is waiting for an incoming call. The phone is connected to a telephone exchange which makes and maintains the connections between telephones. To make a call you lift the handset, at which point the exchange produces a dial tone you can hear from the handset speaker. You use the dial to enter the digits of the number of the phone you want to ring. After you have dialled the last digit of the number, the exchange connects to the destination phone and makes it ring. When the receiver of the destination phone is picked up, the exchange connects the microphone and speakers of the phones together so that the phone users can have a conversation. When either handset is placed back on the phone, the exchange ends the call. The software inside the Raspberry Pi emulates this process.

YOU'LL NEED

- **An old-style telephone with a dial.** The author used a 746 model he picked up in a second-hand shop. The author thinks the phone should be red, like the original 'batphone', but the software will work with other colours
- **A Raspberry Pi Zero 2**
- **2 × power switches for the bell.** The author used one with dual D4184 MOSFETs which can be driven by the GPIO signals from the Raspberry Pi
- **2 × 1N4007 1A 1000 V silicon rectifier diodes**
- **A power converter to convert 12 volts to 35 volts for the bell**
- **A power converter to convert 12 volts to 5 volts for the Raspberry Pi**
- **A 12-volt power supply for the above**
- **The author used a Raspberry Pi GPIO breakout board to mount all the components**
- **A USB audio adapter and micro USB 'on-the-go' cable to connect it**

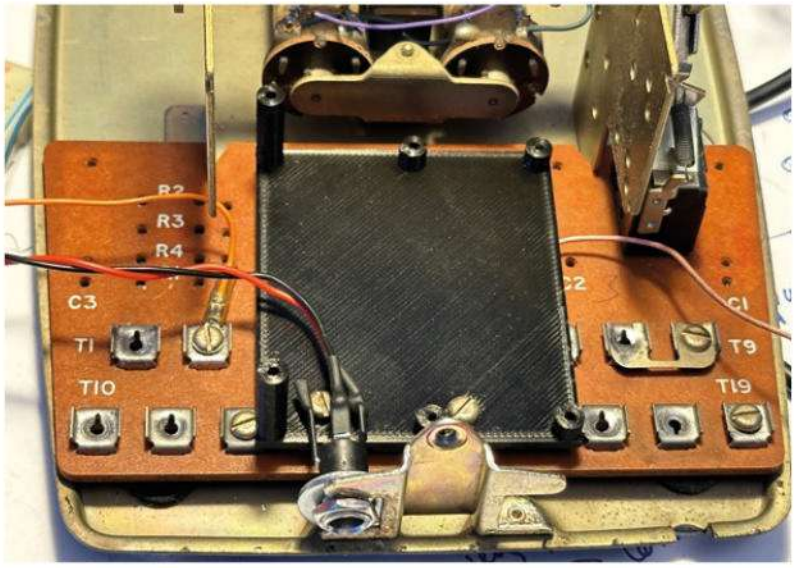


Figure 3
The remaining two connections (orange wire and pink wire) are for the switch that the Raspberry Pi will use to detect when the handset is lifted. The switch that detects the handset remains on the circuit board

flag after each ding and will stop ringing if this flag becomes false.

```
async ukRing() {
    while (this.ringing) {
        await this.repeatRing(10);
        await this.delay(100);
        await this.repeatRing(10);
        await this.delay(1400);
    }
}
```

The **ukRing** method produces the characteristic 'brring-brring' sound of a UK telephone, which is two rings separated by a silence of around one and a half seconds. The length of each ring and the intervals between them were also determined by trial and error. In fact, the completion of this project seems to have been accomplished with a lot of trial and an awful lot of error.

MULTITHREADING WITH AWAIT

The statements in the **ding** and **ukRing** methods above use the JavaScript **await** keyword to ensure that the phone application can still respond to events (for example, the handset being lifted) while the bell is ringing. When a JavaScript program reaches an **await** in a method or function, it creates another thread to run from that point and returns, allowing the calling function to continue. In other words, in the **ding** method above, the first call of **delay** (which pauses for 25 milliseconds) will not pause execution of the program that called **ding**. Instead, the **ding** function will return to the caller at that point. The **ding** function is flagged as async, which means that it returns a **Promise** object to be used by the caller to trigger other actions when the ding has completed. The **Promise** can be given event methods to be called when the promise is resolved (i.e. when the ding has finished). This form of multithreading is very flexible.

If it seems confusing, imagine that you could create new versions of yourself at will. If you need to queue to buy something, you could create a new version of yourself, put that in the queue, and then go about the rest of your business. At some point the 'new you' will reach the head of the queue, get what you want, and then call you and say it has finished before vanishing in a puff of smoke. That's how awaits and promises work.

Figure 5, overleaf, shows the contacts inside the dial. The white plastic cog in the centre has ten teeth for each number. The plastic follower at the bottom right-hand side of the dial swings out of the way when the user is moving the dial clockwise and then engages with the metal contacts when the dial is returning, opening and closing a contact as it does. The further the dial is turned, the more teeth will hit the follower, and the more pulses will be sent. It is an ingenious piece of mechanical design. The contacts near to the centre of the cog in the centre are closed when the user moves the dial from its 'home' position so that the exchange can be told that the user is dialling a number.

Figure 6, overleaf, shows the signals produced by the dial when the user dials a three. The telephone program in the Raspberry Pi must read these signals to obtain this value so that it can be used to control the phone. The decoding software uses functions which are bound to events generated when the dial signals change state.

```
startDialing(){
    this.pulseCount = 0;
    this.dialing = true;
}
```

READ THAT DIAL

The telephone dial is used to enter phone numbers. The user puts their finger into the required number hole (see **Figure 1**) and turns the dial clockwise until their finger hits the metal stop at the bottom of the dial. The dial is spring-loaded. When the user releases the dial, it turns back to its home position. As the dial rotates back to its home position, it sends a series of 'dial pulses' to the exchange. If a larger number is entered, the dial will rotate further. Telephone numbers contain a particular number of digits; once the last digit has been entered, the exchange will connect the call.

QUICK TIP

There is a useful guide to old telephones here: **hsmag.cc/phoneT746**. This includes some illustrations that the author thinks would look great on T-shirts.

```
dialPulse(){
    if(this.dialing){
        this.pulseCount++;
    }
}

endDialing(){
    console.log(`Dialed a:${this.pulseCount}`);
    this.owner.numberDialed(this.pulseCount);
    this.dialing = false;
}
```

The **startDialing** method above is called when the software detects a rising edge (a change from low to high) on the 'Dial Active' signal. It sets **pulseCount** to zero and sets dialling to true, which indicates that a number is being dialled. The **dialPulse** method is called on the rising edge of the 'Dial Pulse' signal. It checks to see if a number is being dialled and increments **pulseCount** if it is. The **endDialing** method is called when there is a falling edge on the 'Dial Active' signal. This calls the owner of the dial and delivers the pulse count to the dial owner by calling the function **numberDialed**. It then turns dialling off by setting the dialling flag to false.

> The further the dial is turned, the more teeth will **hit the follower, and the more pulses will be sent**

ADDING AUDIO

The Raspberry Pi Zero in the phone uses a USB audio hardware interface to produce sounds. The output is quite capable of driving the speaker in the handset. The JavaScript program uses the **play-sound** library to play sound files and the eSpeak program to convert text to speech. Presently, the phone doesn't support audio input. This is because the microphone in the telephone is implemented using a little foil box of carbon granules which change in resistance when vibrated by sounds. This change in resistance is used to drive a coil in a transformer to generate the audio signal to be sent over the phone line. This microphone cannot be connected directly to the microphone input on the USB sound interface. The author intends to investigate using the transformer removed from the phone to see if this could create a usable signal. However, the phone is still great fun to use, even if you can't speak into it just yet.

BUILDING THE PHONE

Figure 7, overleaf, shows the completed phone hardware. The two MOSFET switches are mounted on a Raspberry Pi prototyping board which is plugged into the Raspberry Pi Zero. The USB sound interface is plugged into a micro USB 'on-the-go' adapter at the back of the phone. The author was careful with the 35-volt signals, especially after he destroyed a Raspberry Pi Zero by accidentally touching one of the ringer coil terminals with a GPIO input.

PHONE SOFTWARE

The phone software is implemented as several cooperating JavaScript classes, each of which looks after one part of the phone. The **Phone** object acts as a container for these.

```
constructor(owner) {
    this.owner = owner;
    this.ringer = new Ringer();
    this.handsetSwitch = new HandsetSwitch(this);
    this.dial = new Dial(this);
    this.soundOutput = new SoundOutput(this);
    this.speech = new Speech(this);
    this.ringing = false;
    this.ringStart = null;
    this.ringLengthMillis = 10000;
    this.dialing=false;
    this.messages = null;
    setInterval(() => {
        this.update();
        }, 500);
}
```

QUICK TIP

The diodes across the bell coils protect the MOSFET switches from the reverse voltage induced in the bell coil when the bell current is turned off. The diodes are not expensive, but the author has discovered that missing them out can be.

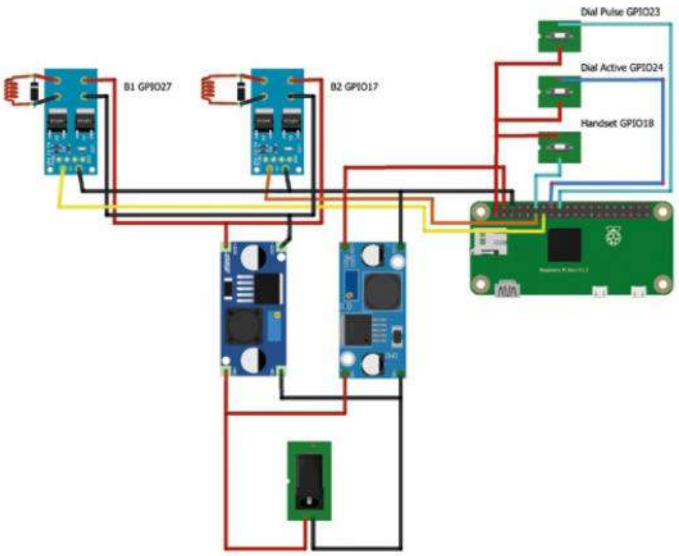


Figure 4 There is also USB audio device plugged into the Raspberry Pi to provide the sound output from the phone

QUICK TIP

The story that the first telephone dials were invented by an undertaker to stop telephone operators (people who connected telephone calls for a living) from learning about local deaths and passing on the details to his competition is not true. Almon Brown Strowger was an undertaker, but he just wanted to improve the accuracy of his telephone connections.

The code above is the constructor for the **Phone** class. It creates all the different phone component objects and speech and sound output services, sets up some initial values, and then starts an update timer ticking which can do things such as time out the ringer. The component objects trigger actions in the phone by calling methods in the **Phone** instance. For example, the **Phone** class contains a method called **numberDialed** which is called by the **Dial** class.

```
numberDialed(number){
    if(this.handsetSwitch.handsetOffPhone()){
        this.delay(600).then(()=>
        {
            switch (number){
                case 1:
                    this.startRinging();
                    break;
                case 2:
                    this.randomCall();
                    break;
            }
        });
    }
}
```

The code above shows **numberDialed**. It waits for around half a second to simulate the exchange connection delay, and then if a 1 was dialled, the phone starts ringing. If 2 was dialled, the handset makes a random call.

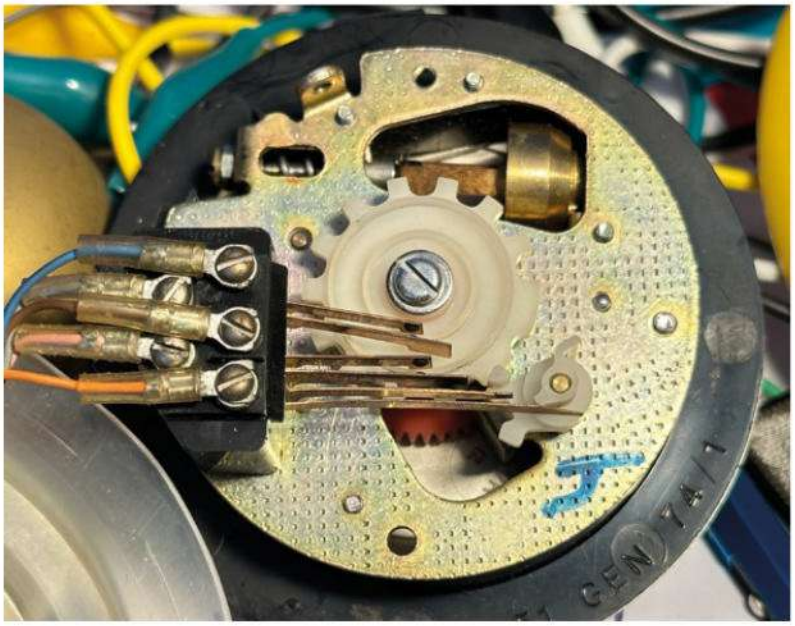


Figure 5
The little brass cup at the top right-hand side of the dial contains a mechanical regulator to limit the speed at which the dial turns back

```
randomMessages = [
    "I know what you did last summer.",
    "Is that you, Boris?",
    "Look out of the window.",
    "They are on to you.",
    "Look behind you."
    ];

randomCall(){
    let messageDelayMillis = this.
getRandom(2000,5000);
    this.delay(messageDelayMillis).then(()=>{
        let messageNo = this.getRandom(0,this.
randomMessages.length);
        this.acceptMessage(this.
randomMessages[messageNo]);
    });
}
```

> It is fun to play with the telephone itself, but **the phone is even more fun when controlled remotely**

The **randomCall** method waits a random time between 2 and 5 seconds, picks a message from the **randomMessages** array, and then calls the **acceptMessage** to play it.

```
acceptMessage(message){
    this.message = message;
    this.startRinging();
}
```

The **acceptMessage** stores the message in the **Phone** class and then starts the phone ringing. When the handset is picked up, the message is played.

```
handsetPickedUp(){
    if(this.ringer.ringing){
        this.stopRinging();
        if(this.message){
            this.delay(1000).then(()=> {
                console.log(`Saying
message:${this.message}`);
                this.speech.say(this.message);
                this.message = null;
            });
        }
    }
```

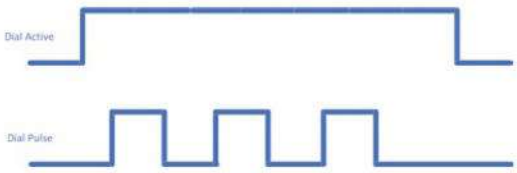


Figure 6
The 'inout' library is used to allow a Raspberry Pi JavaScript application to interact with the GPIO pins

```
    else{
        this.ringer.ding().then(()=>
        {
            this.doDial();
        });
    };
}
```

The **handsetPickedUp** method is called when the handset is picked up. Which just goes to show how good the author is at picking names for methods. If the ringer is ringing, it turns it off and then checks to see if the phone has a message to say. If it does, the message is spoken after a short delay to give the user time to get the handset to their ear. If the ringer is not ringing when the handset is picked up, it calls the **doDial** method to start dialling a number.

```
doDial(){

    if(this.dialing){
        console.log("doDial called when already
dialing");
        return;
    }

    console.log("Dialing");

    this.dialing = true;

    this.soundOutput.playFile('dialTone');
}
```

The **doDial** method starts the dialling process. It sets a flag to indicate that the phone is dialling (having first checked that the phone is not already dialling) and then plays the dial tone sound effect.

From the code above, you can see that the phone works as a series of methods which are called in response to events and modify values held in the phone object. You can use this to make the phone respond to dialled numbers in any way you like.

Figure 7
The two voltage converters are mounted on holders which are then stuck to the base of the phone. You can see them on each side just above the dial

More advanced phone code could assemble longer sequences of dialled numbers. The author thinks it might be fun to create a 'Phone powered' mystery game where the phone rings every now and then and invites the player to perform an action and dial responses to move through the game.

MAKING A TELEPHONE INTO A WEB SERVER

It is fun to play with the telephone itself, but the phone is even more fun when controlled remotely via the local network. The RedServer.js program hosts a website which can be used to remotely control the phone.

The Raspberry Pi in the telephone hosts the web page shown in **Figure 8**, overleaf, on your local home network. The author has set the Raspberry Pi machine name as 'theredphone' and the site is hosted on port 3000. This means the site can be found on a home network as **theredphone.local:3000**. The web server is powered by Express, a popular library for hosting web pages.

The Express library allows you to create 'routes' which are pieces of code that are executed when the client browser is used to access a particular web address.

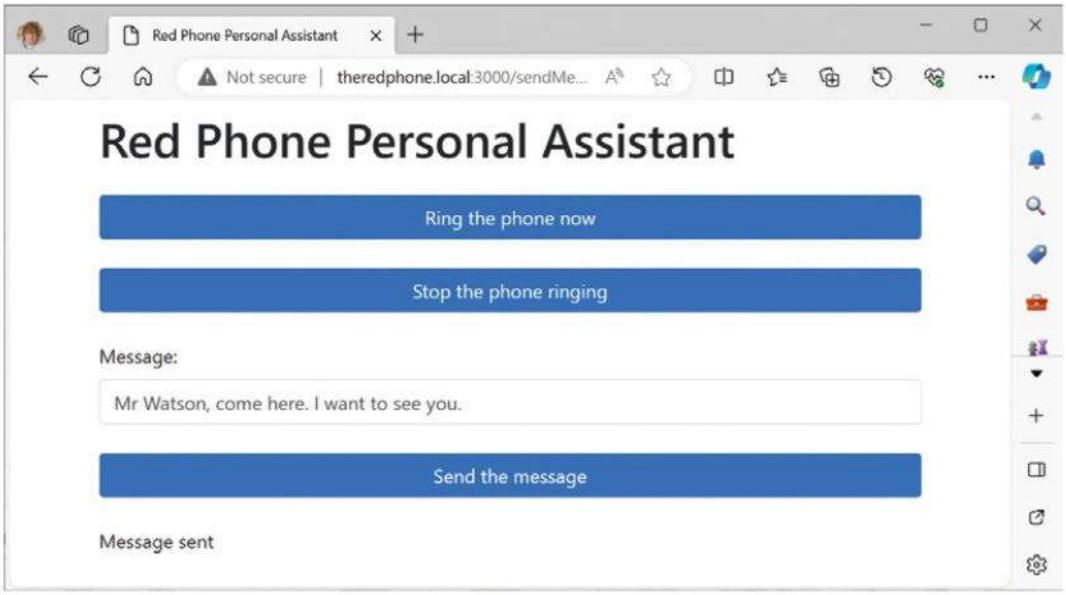


Figure 8
When 'Send the message' is clicked, the phone will ring. When the receiver is picked up, the phone will speak the message that was entered into the form

```
app.get('/', (req, res) => {

  res.render("index.ejs", {message:''});
});
```

The code above is performed when the user browses the site. The Raspberry Pi in the telephone web server uses the **ejs** library which allows us to create web pages that contain JavaScript elements. When the root is accessed, the server displays the index page you can see in **Figure 8** with a message value of an empty string. The page layout is described in the **index.ejs** file:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>Red Phone Personal Assistant</title>
</head>

<body>
        <h1 class="mb-4">Red Phone Personal
Assistant</h1>
        <a href="/ring">Ring the phone now</a>
        <a href="/stopRing">Stop the phone
ringing</a>
        <form action="/sendMessage"
method="POST">
            <label for="email">Message:</label>
            <input type="text" id="message"
name="message" required>
            <button type="submit">Send the
message</button>
        </form>
       <p> <%= message %> </p>
```

> It would be interesting to add speech decoding so that **the phones can recognise what the user says**

```
</body>
</html>
```

When the user fills in a message and clicks 'Send the message', the POST action sends the message text back to the `sendMessage` server which runs a handler that gets the message out of the body of the web request and asks the phone to play it.

```
app.post('/sendMessage', (req, res) => {
  phone.acceptMessage(req.body.message);
  res.render('index.ejs', {message:'Message
sent'});
});
```

You can add more remote commands by adding routes to the index page and then creating the JavaScript handlers to deal with them.

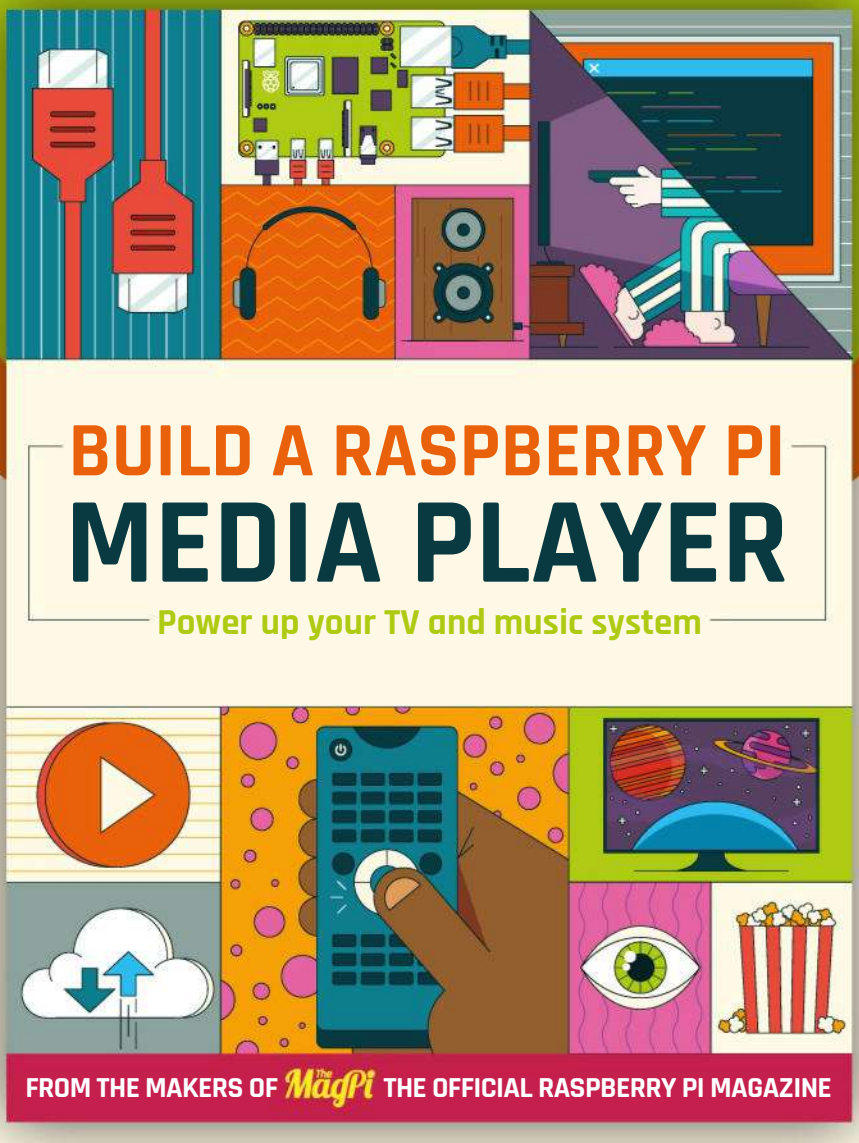
FURTHER DEVELOPMENT

The author is very pleased with the phone and it works well. He has even bought a second one with a view to connecting them together – once he has figured out how to make their microphones work. It would be interesting to add speech decoding so that the phones can recognise what the user says. The Raspberry Pi inside should be able to do this.

QUICK TIP

The code for zero generates ten dial pulses, since sending zero pulses would make the dialling process vulnerable to noise (a noise pulse on the dial active line would be interpreted as dialling the value zero). Remember that in a real telephone, these signals are sent over cables to the telephone exchange.

Your FREE guide to making a smart TV



magpi.cc/mediaplayer

TUTORIAL

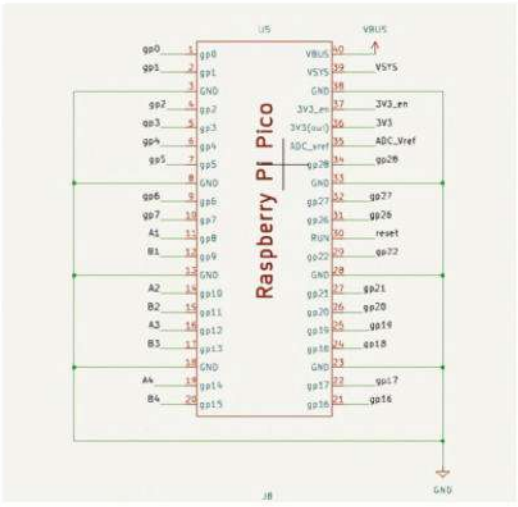
KiCad, mechanical accuracy, and silkscreen features

In this part of the ongoing KiCad series, let's look at some techniques to increase accuracy when aiming to create a PCB to be used as a mechanical structure

Jo Hinchliffe

Jo Hinchliffe is a constant tinkerer and is passionate about all things DIY space. He loves designing and scratch-building both model and high-power rockets, and releases the designs and components as open-source. He also has a shed full of lathes and milling machines and CNC kit!

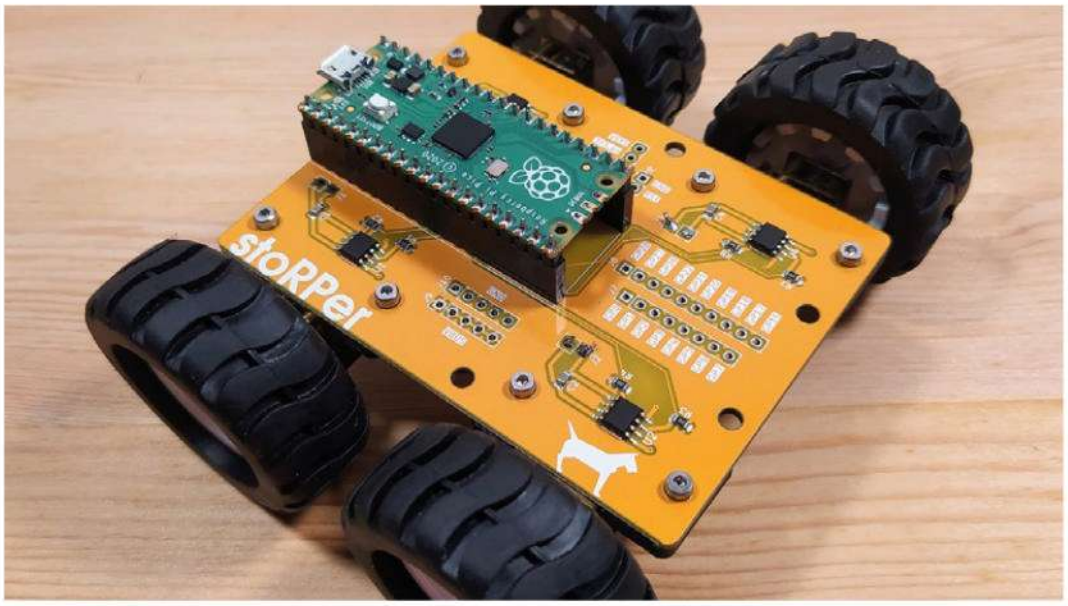
It's increasingly common for projects to incorporate PCBs as a mechanical part of the mechanism. In our last section, we looked at hierarchical sheets and laid out a motor driving circuit that we could copy and paste to add motor drivers to a project. In this part, we are going to create a simple robot rover that we're calling 'stoRPer'. StoRPer is a tongue-in-cheek reference to a favourite childhood toy from the 1980s: the 'Stomper'. The Stomper, by toy company Schaper, was the first ever four-wheel drive electric toy car. Despite no form of remote control, they were great fun to try and build obstacle courses for, or to test on steep gradients. We wanted the reasonable torque and the four-wheel drive aspects of the Stomper but with the addition of a Raspberry Pi Pico to make it a more interesting and controllable platform – so, 'stoRPer' it is. It's designed with all-wheel drive (AWD) so that Mecanum drive systems can be built and experimented with.



Right
A custom Pico symbol has been created, with the majority of the pins broken out

We are going to use Pico as a module on this build and focus on some aspects of particular importance when we are using a PCB as a mechanical part as well as for electronic purposes. The idea for the project is that the PCB will form the chassis of the stoRPer, with the motors being clamped to the PCB chassis using some 3D-printed parts. Therefore, we need to be capable of placing components and general PCB geometry accurately in order for everything to fit together. We'll also look at how we can check our PCB and 3D-printed models will fit together before we print or send the PCB for fabrication.

One of the first jobs is to create a Pico symbol component in the Symbol Editor. We covered creating symbols in the earlier sections of this series, so we won't recap that process too much. We decided not to include the Pico's three debug pins on either the schematic symbol or the PCB footprint. This was partly because, across the different Pico models, they are physically in different positions on the board and also, as we intend to have a Pico mounted onto this project, we can still interact/wire to the debug pins if needed. As such, we laid out a simple 40-pin component in the Symbol Editor and brought it into our Schematic Editor. After quickly connecting all the ground points, we set about connecting four hierarchical sheets, each with an L9110S motor driver



Left
The stoRPer robot prototype using the PCB as its main chassis component

Figure 1
The layout of the L9110S motor driver circuit cloned into four hierarchical sheets

IC-based circuit inside. We covered working with hierarchical sheets in the last section of this series, but you can see the circuit layout in **Figure 1**. Each of the four motor drivers has its own sheet and has two pins broken out. We've connected these sets of pins to the Pico symbol using labels A1, B1, A2, B2, etc. The rest of the Pico's pins are broken out and connected to some multi-pin connectors, ready to be broken out on the PCB.

For the stoRPer project, we've decided to mount the Pico using the through-hole header pads on the Pico rather than the castellated edge connectors.

> The rest of the Pico's pins are broken out **and connected to some multi-pin connectors**

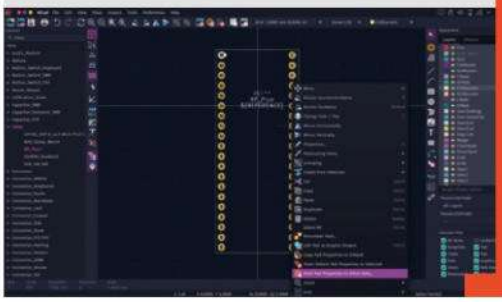
This means that we won't be mounting the Pico flush to the project, but it does mean that the Pico footprint is thinner. We can also choose to use header sockets or not to allow the Pico to be permanently or temporarily mounted to the PCB.

The header pin pads on the Pico lie in a 2.54 mm pitch grid, with the 20 pins on either side being separated by 7*2.54 mm. This makes them easy to lay out – simply add pads on a 2.54 mm grid in the Footprint Editor (**Figure 2**). We also want to be able to place a rectangle on the silkscreen layer that accurately shows the position of the board.

Consulting the Pico documentation, we can find a technical drawing and see that the outer edge of the Pico is 51 mm × 21 mm. We also need to consider the position of this rectangle relative to the pads that we have just created. We can see in the technical drawing, for example, that relative to the centre of the upper left-hand pin (pin 1), the upper-left corner of the Pico is 1.37 mm higher in the Y axis and 1.61 mm over to the left in the X axis. To use this information, we can go back into the Footprint Editor and place our pointer on the grid point in the centre of the pad we placed for pin 1. If we then press the **SPACE** bar, we will set the local origin of the page to be 0,0 at this point. We can check this by looking at the bottom of the screen as we move our pointer, the distance should increase relative to this point. We can then set a user grid to 1 mm spacing and use this grid to draw our 51 mm × 21 mm rectangle.

LOTS OF HOLES

When creating footprints with lots of through-hole pads, KiCad makes it pretty simple: you click to add a pad and then the tool indexes to the next numerical pad for you to place. If you've placed and positioned a lot of pads though, it can be annoying to realise that you need to change an aspect of the pad's properties for all of them. KiCad has you covered, though. As an example, when we made the footprint for a Raspberry Pi Pico and decided that after laying out 40 standard through-hole pads, we wanted to increase the internal hole diameter and the overall outer diameter to increase the size. The Footprint Editor conveniently recognises that this is a common situation and, as such, you can simply change one pad to your desired pad properties and then, with your adjusted single pad highlighted, you can right-click and select 'Push Pad Properties to Other Pads…' to make all compatible pads change to the new characteristics.



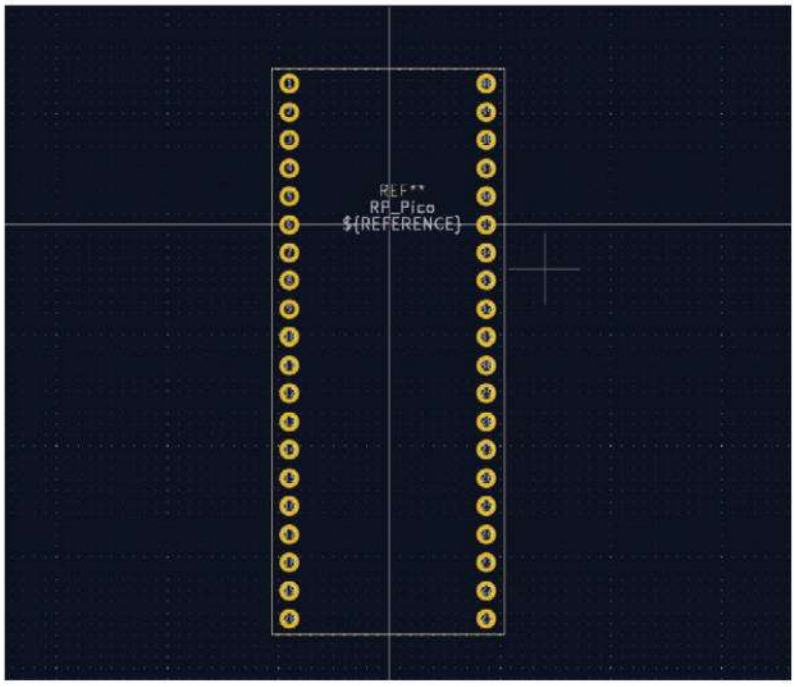


Figure 2
Creating the simple yet accurate Pico footprint

If we then select the rectangle, we can right-click and scroll in the drop-down menu to Positioning Tools > Position Relative To…. Selecting this, we will see a dialog box – in the dialog box, click to select 'Use Local Origin' and then adjust the 'Offset X' and 'Offset Y' by the amounts we derived from the technical drawing (**Figure 3**). Note that, by default, the origin corner of the rectangle is the top left-hand corner. Using this method, you should be able to place items with incredible accuracy.

One thing of note is that despite our stoRPer robot design being relatively simple mechanically – a rectangular PCB – we do want to be able to place footprints accurately within the edge cut area. When designing this and other footprints, it's worth considering where your origin point is in the Footprint Editor and placing the device in a known position relating to it. We opted to place the Pico footprint so that the upper left-hand corner of the silkscreen box depicting the edge of the Pico was the origin point on a 1 mm grid spacing. This meant that later, when we placed a rectangle in the PCB Editor that represented the edge of the PCB, we could place it in a position such that the Pico is dead centre, with the larger box also placed on a 1 mm grid coordinate. After playing with a few test boxes in KiCad, we decided our rectangular chassis dimensions would be 64 mm × 86 mm. We used Inkscape to draw our rectangle as we could then easily add a 2 mm radius to each corner

> "We want to 3D-print some brackets to **clamp the motors into position**"

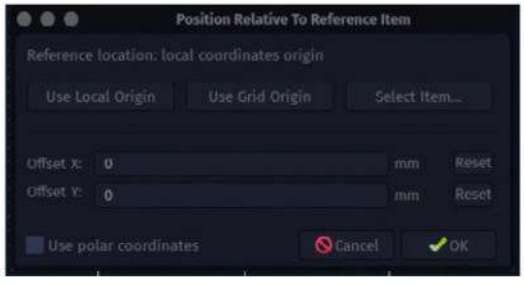
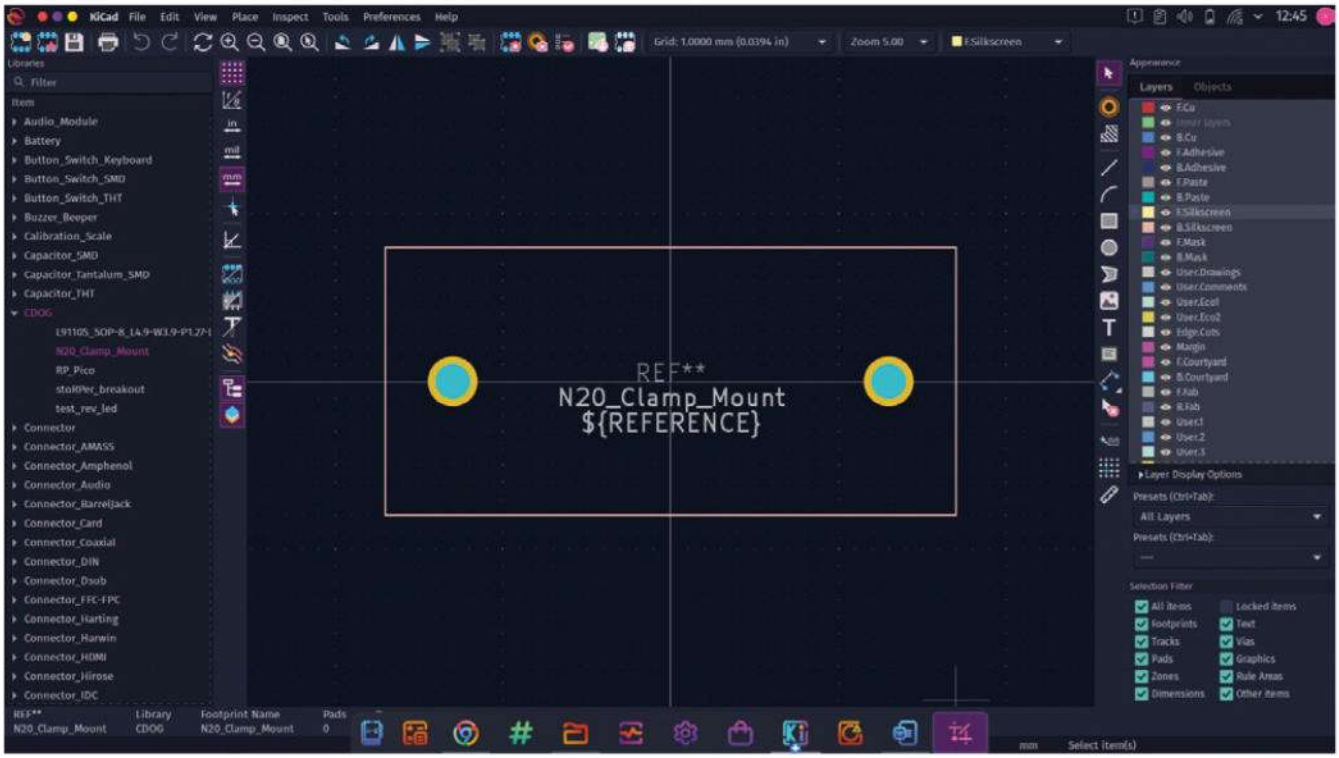


Figure 3
Using the 'Position Relative To…' positioning tool to accurately place objects in the Footprint Editor



of the rectangle. We've again covered importing graphics in an earlier section of this series, but we easily imported the rectangle we drew as an SVG in Inkscape into our edge cuts layer using the File > Import > Graphics function.

With the Pico placed and the PCB edge defined, we need to consider the physical mounts for the motors. We want to use the excellent and available N20-style geared motors, mounting one for each of the four motor driver circuits. We want to 3D-print some brackets to clamp the motors into position, so we need to leave some space for the 3D print material around the motor, and we need to take this into account when creating a footprint for the motor mount. After some consideration, we created a custom footprint which consisted of two non-plated through-hole (NPTH) mechanical pads placed in-line. These were placed at a distance between centres of 26 mm, placed on a 1 mm grid spacing. To place an NPTH mechanical hole, you place a regular pad and then press **E** to change the pad type in the Pad Properties dialog. We set each NPTH hole to 2.1 mm internal diameter to create clearance for a small M2 bolt. To finish the footprint for the N20 motor mount clamps, we added a silkscreen rectangle set to the dimensions of the base of our 3D-printable mount design (**Figure 4**). Note that this is the first time in the series that we have placed extra components which aren't connected to anything or included in the schematic in the PCB Editor. To do this, we click

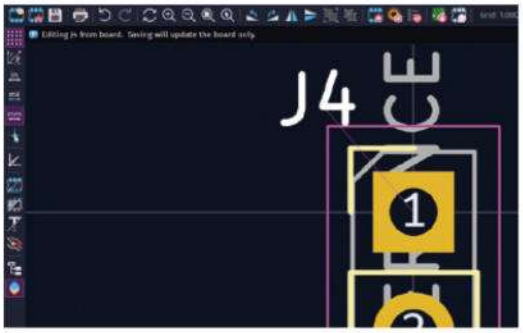


Figure 4
The mechanical footprint that will mount the N20 motor and clamp

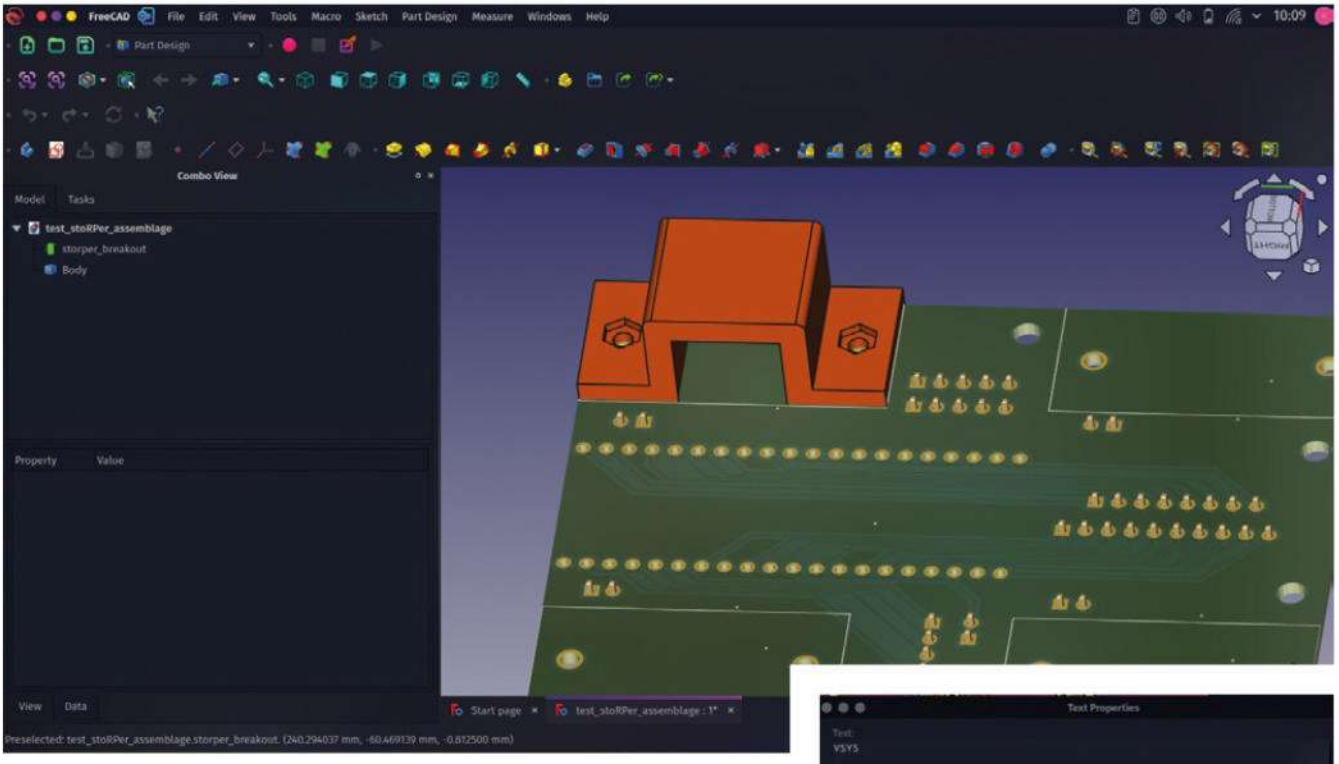
Figure 5
Editing a footprint with the component selected and opened in the Footprint Editor from the PCB Editor gives the option of only editing that individual instance of the footprint

the 'Add a footprint' tool icon and select a footprint in a similar manner to how we would place a symbol in a schematic.

Adding and removing text-based elements to a silkscreen layer is reasonably straightforward in KiCad 7. On more technical PCBs, as opposed to artistic PCBs, we often lay out our PCB design with little regard for the silkscreen and then sort the silkscreen layer out later in the development. Often, one of the first tasks is to remove unwanted elements on the silkscreen that have been automatically placed by the use of default library footprints. We can select the correct silkscreen layer, often the front silkscreen 'F.Silkscreen', and for items such as footprint reference annotation, we can simply left-click to select them, then move them or press the **DELETE** key to remove the item. It's common for this reference to not be placed →

QUICK TIP

As the stoRPer design evolved, we used simple rectangular boxes drawn in KiCad on either the F.Silkscreen or the User.Comments layer as guides and visual aids.



Above
The combination of KiCAD and FreeCAD make a great open source toolchain

FREE BOOK

In the free-to-download book *FreeCAD for Makers* from Raspberry Pi Press, we looked extensively at the use of the KiCad StepUp workbench which enables and simplifies importing KiCad projects as 3D objects into FreeCAD as well as the creation of 3D components for inclusion into KiCad's 3D PCB viewer. It's an incredibly powerful suite of tools and is definitely worth exploring. For this project, however, we just wanted to check if the motor clamp we had created in FreeCAD would fit our PCB chassis. You can use File > Export and select the 'STEP…' option to export a STEP file which can be imported into FreeCAD; however, this will lack the details of the copper layers and silkscreen which you might need to see to check if mechanical components cover aspects of your PCB design. One simple approach that solves this is to export a WRL file. WRL files are file types often used by assets destined for use in virtual reality, but they have the advantage in KiCad that a WRL export contains all the visual details of your PCB. We used File > Export > 'VRML…' to export a WRL file, and then we used File > Import in a new document in FreeCAD to import the file. We'd made a simple N20 clamp component which had 2mm radius corners on two corners matching our PCB and N20 motor clamp footprint. While we could have used an Assembly workbench, such as A2plus in FreeCAD, to constrain the clamp in position, for a simple check, we can move the part into alignment to visually check how it looks.

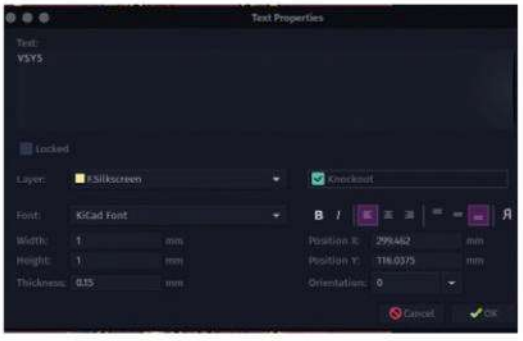


Figure 6
The Text Properties dialog where you can set text features, including the new 'Knockout' feature

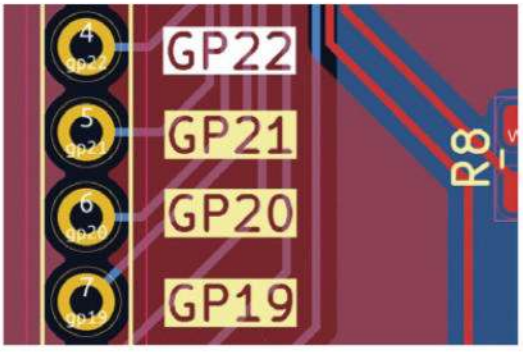


Figure 7
A new feature in KiCad 7 is the ability to add knockout text items, where the text is subtracted from a small block on the silkscreen layer

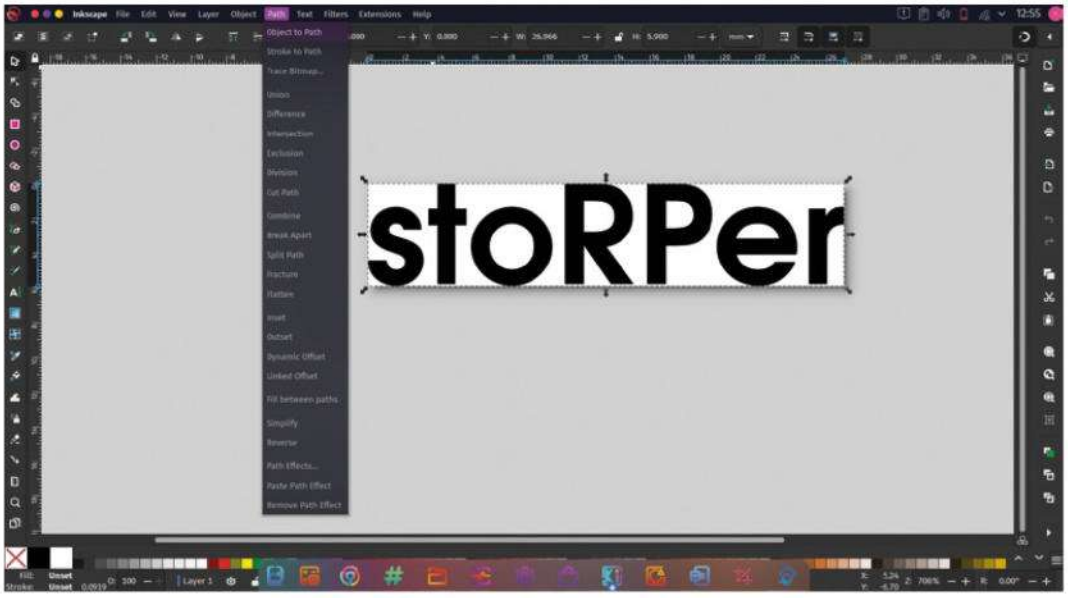


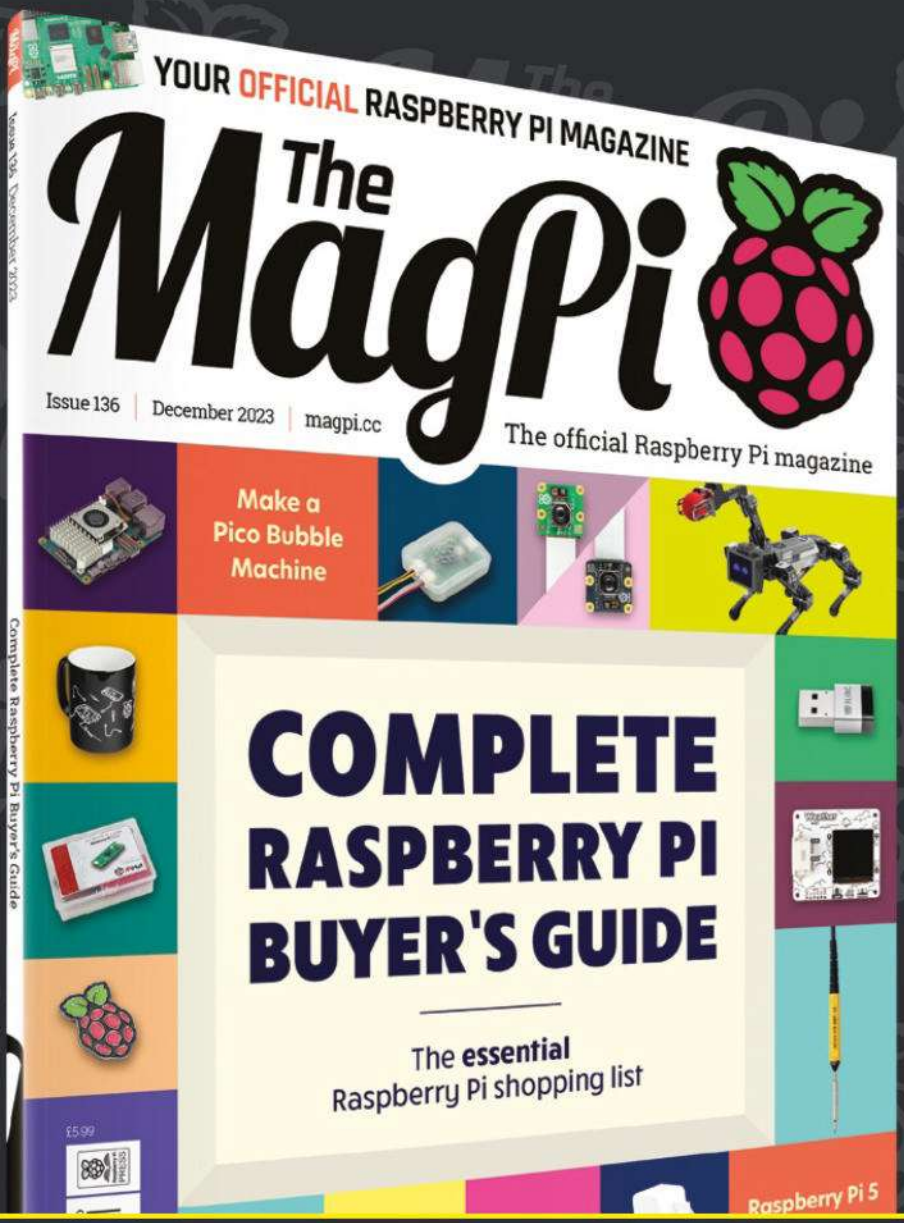
Figure 8
Converting a text object to a path in Inkscape ready for import into KiCad

optimally and may sit under or across other parts and components. The annotated reference is formed from both the automatic annotation of the schematic during the footprint association process and the type of component it is, so R for resistor, C for capacitor, J for connector, U for IC, etc. As they replace the placeholder Ref* designator, they are independent of the main footprint design and, as such, can be removed with ease. If, when tidying the PCB design, you want to move a part of the silkscreen design of a footprint, you will need to edit that in the Footprint Editor. KiCad makes it easy to edit the footprint and apply the changes just to the individual footprint within this project rather than pushing the changes to the global footprint library. With a target footprint selected in your PCB, press **CONTROL** and **E** to open the footprint in the Footprint Editor. You should see the footprint in the editor with a message in the upper left-hand corner of the window that reads 'Editing J4 from board. Saving will update the board only', where 'J4' will be the reference of whatever footprint you have opened (**Figure 5**). You can now make any changes to the footprint that you require, including deletion or changes to the graphical silkscreen elements.

Of course, often we want to add text-based components to our board designs. Again, KiCad makes this pretty straightforward. We can simply click the 'Add a text item' tool icon and then left-click in the PCB design. The 'Text Properties' dialog is pretty straightforward and we can insert text, make changes to the font and size as well as change the orientation of text. One interesting new addition to KiCad 7 is the 'Knockout' option (**Figure 6**). If you input some text into the Text Properties dialog and click the Knockout checkbox, then the text will be created as a solid silkscreen block with the text removed. It's a great effect, looks smart, and is very readable – a welcome new feature (**Figure 7**).

Finally on adding text, sometimes you might like to add text to the silkscreen layer as a graphic rather than directly as text. We've briefly looked at importing graphics before for either creating edge cuts geometry or for importing logo graphics from Inkscape. One notable point is that if you use the text creation tools in Inkscape and then directly try to load them as a graphic element, this will fail as KiCad SVG import doesn't recognise the text elements. This is pretty easy to rectify. As an example, we created our stoRPer text in Inkscape and then, with the text object selected, we click Path > Object to Path (**Figure 8**). As usual, we edit the document properties in Inkscape so that the document is the size of the text object – we then save the file as a standard SVG. In the PCB Editor, we then select File > Import > Graphics to import the file, ensuring to select the correct 'F.Silkscreen' as the graphic layer. The text graphic then imports correctly and can be placed in the design where required.

DON'T MISS THE **BRAND NEW** ISSUE!



SUBSCRIBE FOR JUST £10!

- **THREE!** issues of The MagPi
- **FREE!** Raspberry Pi Pico W
- **FREE!** delivery to your door

+ FREE RASPBERRY PI PICO W*

Three issues and free Pico W for £10 is a UK-only offer. Free Pico W is included with a 12-month subscription in USA, Europe and Rest of World. Not included with renewals. Offer subject to change or withdrawal at any time.

magpi.cc/subscribe

HackSpace
TECHNOLOGY IN YOUR HANDS

FIELD TEST

HACK | MAKE | BUILD | CREATE

Hacker gear poked, prodded, taken apart, and investigated

ONLY THE BEST

Retro arcade roundup

Build your video games!

By **Marc de Vinck** @devinck

As soon as I heard about the Raspberry Pi 5, my first thought was, 'retro arcade'! With the performance boosts and long history of support for classic arcade games, it seems like a great combo. I fully understand that you can do a lot more with a Raspberry Pi 5, but it's hard to say you could have more fun!

Custom arcade builds are always a popular project. I think a lot has to do with the demographic of the Raspberry Pi community, but I also think it's because it's one of those projects that you can easily succeed in accomplishing, and it has a long-term fun factor. It doesn't always have to be a full-size arcade in your house, either. A simple shoebox-size enclosure is all you need to get a Raspberry Pi hooked up to your TV and have a ton of fun on your next game night.

In this roundup, I'll be looking at a few boards and accessories to get you inspired to build your own retro arcade – mostly Raspberry Pi-based, but not all. And if you are clamouring to get your hands on a Raspberry Pi 5, you're going to need a project to kill some time while you wait for it to ship, so why not build an arcade?!

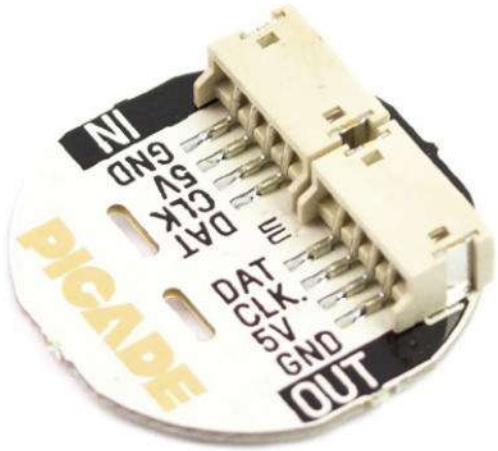
Arcade Parts Kit vs Picade Plasma Kit

PIMORONI $26.35 | pimoroni.com

PIMORONI $41.91 | pimoroni.com

If you're just getting started building an arcade cabinet, the Arcade Parts Kit by Pimoroni is a very convenient and affordable starting point. Pick up one set for a simple one-player game build, or a second set for some two-player action.

Each kit comes with the proper eight-way gated joystick for that traditional arcade feel. The kit also includes ten 30-millimetre push-fit arcade buttons, including four black, two yellow, two pink, and two blue. You also get a very handy matching wiring loom for hooking it all up quickly and easily. And speaking of hooking it up, don't forget to pick up a Player X USB Game Controller, also available at Pimoroni, for just $12 to make connecting all of this to your Raspberry Pi a breeze.



Picade Plasma Kit from Pimoroni is an interesting and colourful addition to your arcade build. Why settle for those ordinary plastic buttons when you can swap them out with the included crystal-clear buttons, and included custom plasma PCBs to add a rainbow of colour to your build? Each PCB fits behind the clear button and features an APA102 addressable RGB LED. Now your buttons can be any colour you want!

You have a choice of a six- or ten-button kit. Whichever you choose, you'll get everything you need for a simple replacement of your old buttons, including 30-millimetre push-fit arcade buttons, the custom-designed plasma PCB, and the required wiring to hook it all up. Check out the Pimoroni website for more information about this fun kit, along with notes on how to hook it up to your Raspberry Pi.

VERDICT

Arcade Parts Kit

A great collection for a good price.

10/10

Picade Plasma Kit

Add some colour to your build.

8/10

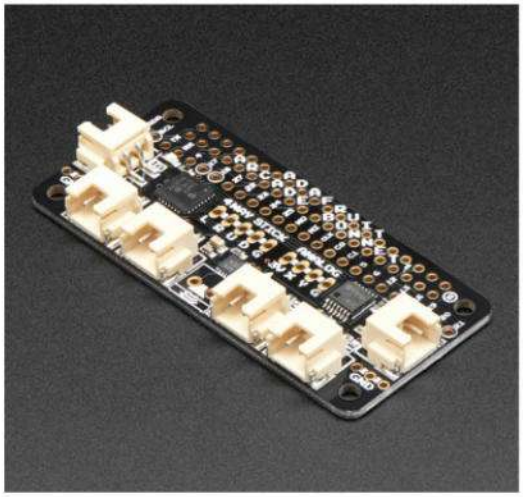
BEST OF BREED

Adafruit Arcade Bonnet for Raspberry Pi with JST Connectors

ADAFRUIT $14.95 | adafruit.com

Yes, I have covered this product before, but when I think of getting started with building a retro arcade system, I always think of the Adafruit Arcade Bonnet for Raspberry Pi. Mostly because I have recommended it to so many people when first starting out with arcade building. No, a Raspberry Pi Zero isn't the best Raspberry Pi for retro gaming, but it runs a lot of the classics perfectly, and it's also incredibly inexpensive.

The Bonnet is the same size as the Raspberry Pi Zero, making it perfect for small builds. And because of all the on-board JST connectors, and with the appropriate wiring harness also available, you can be up and running quickly, and without any soldering. So, if you want a simple and affordable arcade build, be sure to check the Adafruit product page.



VERDICT

Adafruit Arcade Bonnet for Raspberry Pi with JST Connectors

A great addition to your Raspberry Pi.

9/10

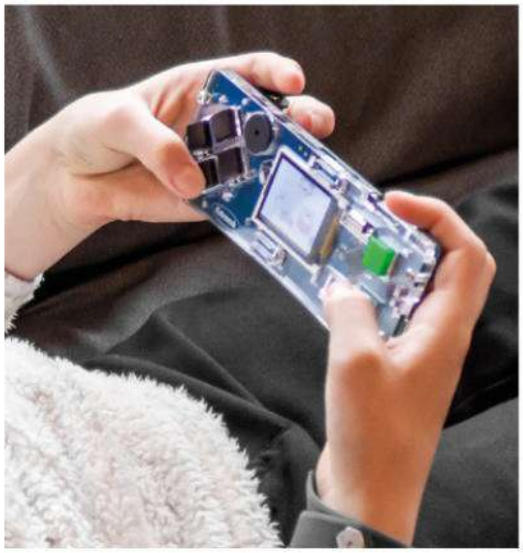
SMALL ARCADE JOYSTICK

ADAFRUIT $14.95 | adafruit.com

I think using a proper-feeling joystick is imperative for any retro arcade build. You'll want that gated and 'clicky' feel that you don't get from modern joysticks. This one, from Adafruit, is a perfect retro-style eight-way gated joystick. It will give you that classic feel and sound thanks to its old-school build. Wiring and usage are easy, since it's basically four buttons connected mechanically to the joystick. No modern potentiometers or encoders here! And that's exactly the point!

ARCADE for MakeCode Arcade

PIMORONI $41.28 | pimoroni.com



Not everything in this roundup is Raspberry Pi-related. The ARCADE for MakeCode Arcade, available at Pimoroni, is a great way to get started building your own handheld arcade games. If you are new to electronics, or even new to programming, but still want to build fun games, then this is a great place to start.

You'll use the Make Code Block Editor to build your games. You can start from scratch or download tons of games available online and modify them how you like. You can also write games in JavaScript, but that's a lot more difficult for beginners. Under the hood is an Atmel SAMD51J19A which controls the 160×128 LCD screen, buttons, haptic feedback motor, and piezo speaker. All you need to do is add 3 × AA batteries and you'll be building or modifying your own games in no time. If you know anyone looking to jump into programming games, this is a great place to start.

VERDICT

ARCADE for MakeCode Arcade

Everything to get started building simple games.

9/10

BEST OF BREED

Pico Display Pack 2.0

PIMORONI $20 | pimoroni.com

The Pico Display Pack is designed to be a simple way to create a user interface device with your Pico. But creating a navigation system for your app or project isn't the only possibility, although it does a great job at that task! Yes, you could also make some simple games! You've got a 320×240 display and four buttons (think X, Y and A, B). What else do you need?

The Pico is very capable of running the graphics and some retro game code. The Display Pack comes fully assembled – no soldering required. Just add your Pico, with headers, and you'll be on your way to programming with this fun add-on. Head over to the Pimoroni website for a lot more information about using a Raspberry Pi Pico and the PicoGraphics function reference to get up and running.

VERDICT

Pico Display Pack 2.0

Perfect for some simple games.

10/10

Arcade Cabinet Kit

GAMEROOMSOLUTIONS $649 | gameroomsolutions.com

If you're still holding out for a Raspberry Pi 5 retro gaming system, why not start on the enclosure? It's one thing to get your Raspberry Pi up and running, but it's a whole other set of skills to house your arcade system in a beautiful enclosure. And nothing looks better than a full-size arcade cabinet.

Not everybody has enough tools or space to build a full-size arcade cabinet. But if you have the room in your house, you can still get your hands on a customizable, full-size arcade cabinet. These kits by Game Room Solutions offer a variety of shapes and customizability. And best of all, you don't need any special tools to assemble them. You'll still have to supply your own electronics, but in most situations, that's the easy part. I managed to fit two of these in my house, and they are always a big hit with visitors.

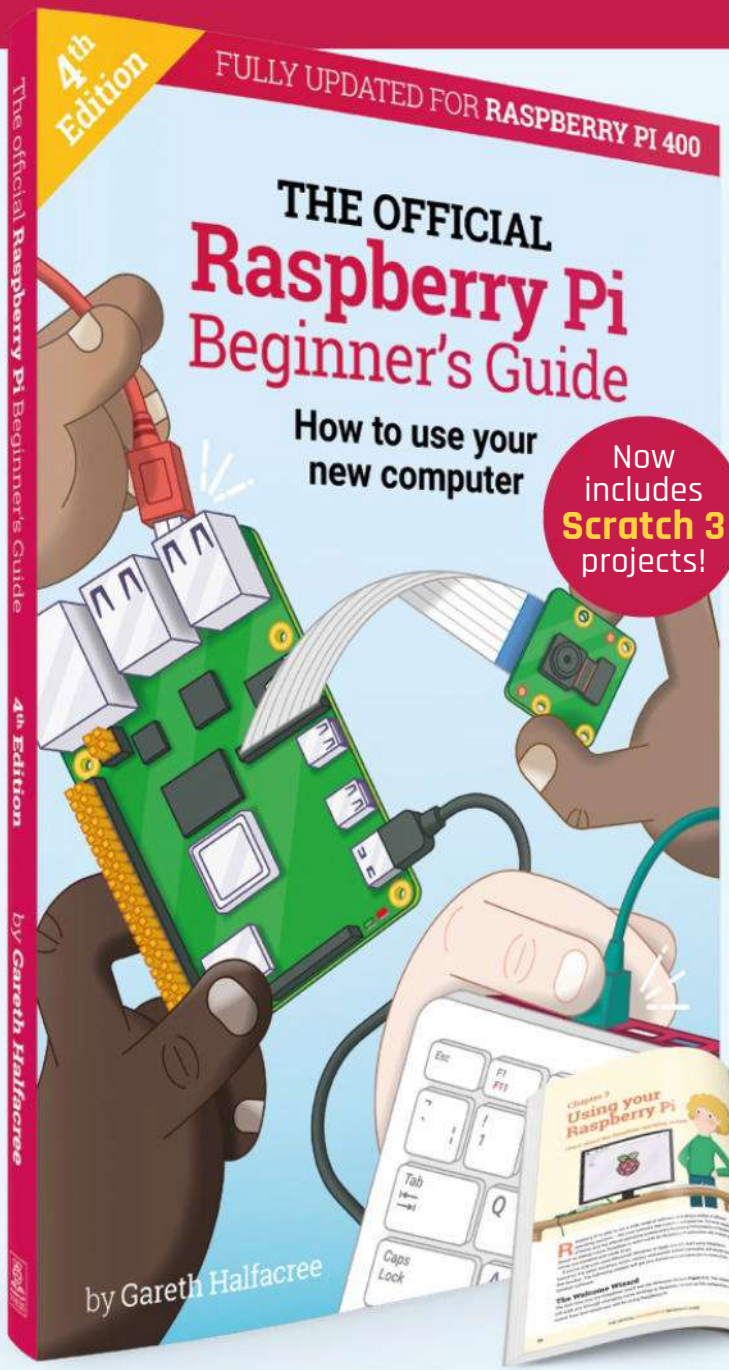
VERDICT

Arcade Cabinet Kit

A good solution if you want to save time.

9/10

THE OFFICIAL Raspberry Pi Beginner's Guide



The only guide you need to get started with Raspberry Pi

Inside:

- Learn how to set up your Raspberry Pi, install an operating system, and start using it
- Follow step-by-step guides to code your own animations and games, using both the Scratch 3 and Python languages
- Create amazing projects by connecting electronic components to Raspberry Pi's GPIO pins

Plus much, much more!

Just £10

Buy online: magpi.cc/BGbook

REVIEW

Metro M7 with Airlift

Who needs efficient code when you've got more clock cycles?

ADAFRUIT ◈ **$29.95** | **hsmag.cc/metrom7**

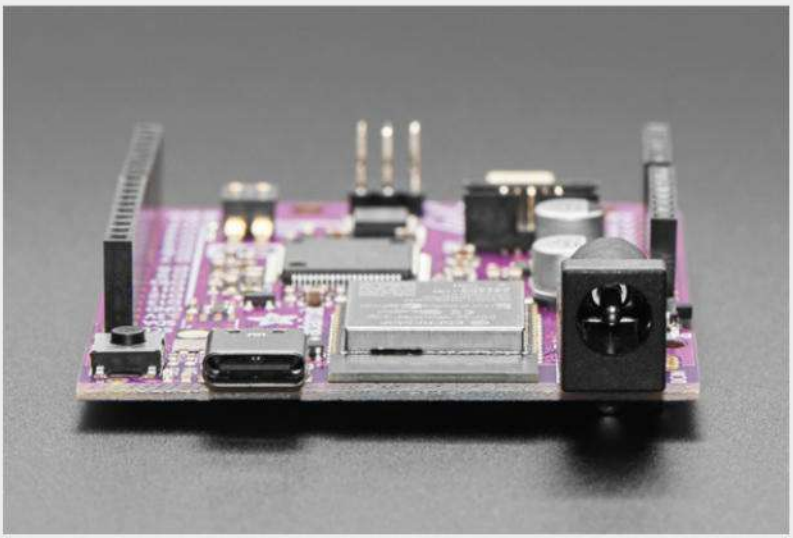
By Ben Everard @ben_everard

The NXP iMX RT1011 that sits at the heart of the Metro M7 is, frankly, a ridiculously powerful microcontroller. It's based on the Arm Cortex-M7 core and runs at 500MHz.

Twenty-two of the microcontroller's GPIO pins are broken out in the classic Uno (what Adafruit calls Metro) style. This means that there's already an ecosystem of shields that can go on top to provide additional hardware, though the majority of these shields come with support for the Arduino programming language rather than CircuitPython, and many are 5V, while this board runs at 3V. If you're planning on using this with a third-party shield, make sure they will work together, not just physically fit.

Hardware doesn't have to be slotted on top, though. The M7 Metro also comes equipped with a Qwiic port for attaching I2C hardware – there's a huge range available from Adafruit and other suppliers.

Below ◈
The ESP32 Wi-Fi module includes an on-board antenna



Alongside the main powerful microcontroller, there's a second microcontroller – an ESP32 that's used for wireless networking. In theory, this can do both Wi-Fi and Bluetooth Low Energy, but at the moment, there's only support for Wi-Fi. Adafruit calls this setup, using a secondary ESP32, Airlift.

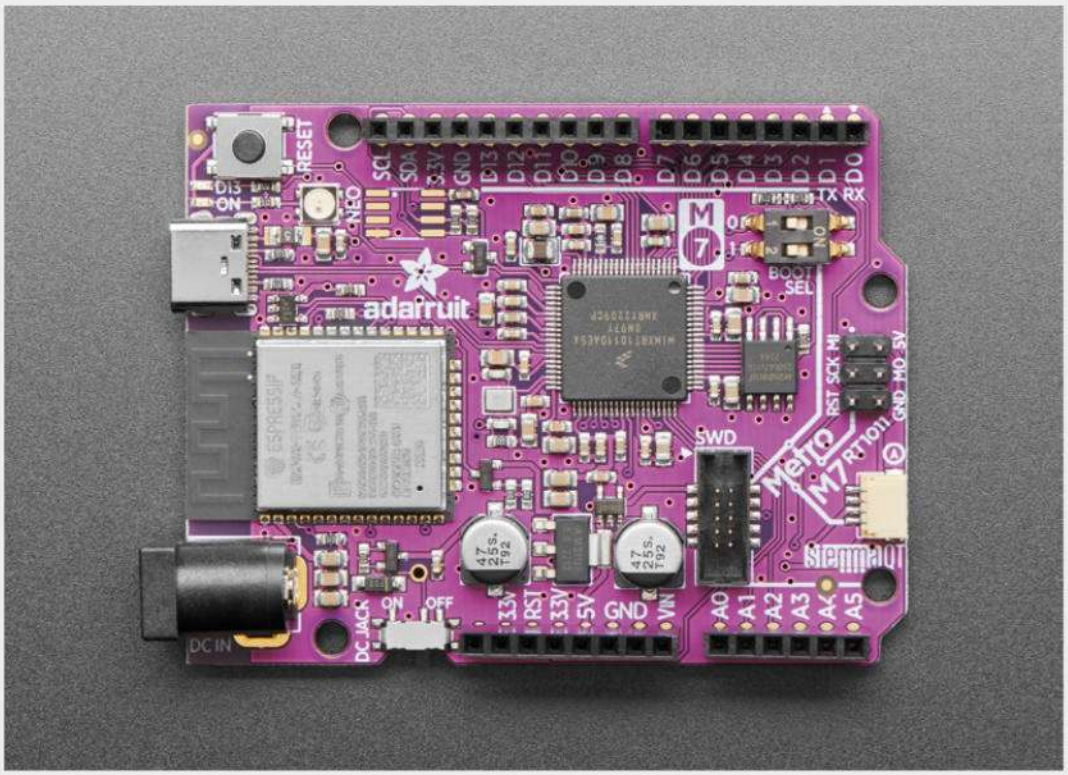
The Airlift networking setup offloads most of the work onto the secondary microcontroller. This means that your main processor isn't burdened with the various issues of keeping connected and shuffling data in and out. That's perhaps less of an issue on this beast of a processor than on some others, but it does mean that your code's performance should be far more predictable.

We tested the Metro M7 Airlift that includes wireless connectivity and costs $29.95, but there's also a version without wireless (and with an SD card port) that comes in at $19.95.

Perhaps the most unusual thing about this board is that – unlike almost all of Adafruit's other boards – you can't use it with the Arduino IDE. You can program it with CircuitPython or the MCUXpresso IDE created by NXP (the microcontroller's designers). For most people, that's likely to mean that this is a CircuitPython board.

IN USE

We tested this out with some audio code. Not so long ago, we were pretty happy if we could make a microcontroller go beep while also doing something else. With this, we were able to play ten WAV files and dynamically adjust the volume simultaneously, and make it Wi-Fi accessible. What's more, we were able to do all of this in Python. Some of this is, of course, down to improvements in hobbyist microcontroller software over the years, but it's also due to the fact that this is almost as powerful as the PC we used to use to program microcontrollers.



Left
With two powerful processors, there's a lot packed onto this board

We also speed-tested the Metro M7 against the Adafruit Grand Central M4 Express and the Metro ESP32-S2. These are two of the fastest CircuitPython boards from Adafruit, running an Arm Cortex-M4 at 120MHz, and a 240MHz Tensilica core, respectively.

We found the M7's performance to be about five to six times faster than the M4 across a range of different areas, including integer and floating-point maths. This is down to both the higher clock speed and the fact that the M7 core can do more computation in each clock cycle. When compared to the ESP32-S2, performance was a bit more varied, but the M7 always came out on top. GPIO access and floating-point arithmetic was about twice as fast, and integer arithmetic was about 4.5 times the speed.

You might think that more computing power is always a good thing, but it does have a drawback. It needs more electrical power to keep it running. Given modern batteries, this is less of a problem than it used to be, but if you need something to run off-grid, you probably want to think a bit about whether you really need this amount of processor power.

The Arm Cortex-M7 is a powerful microcontroller core, but the Metro M7 Airlift isn't the only high-speed Arm Cortex-M7 board, so it's not just a matter of choosing a fast microcontroller – it's a question of whether you want *this* M7 microcontroller. It's reasonably chunky, but whether this is a plus or minus is down to your particular project. Given that this isn't compatible with the Arduino IDE, and that the Uno hasn't been the dominant form factor for microcontroller add-ons for over half a decade, it's unlikely that this form factor is going to be important to you. That said, we're quite fond of this size. It's not too fiddly to work with, but still small enough to fit most spaces, and we prefer socket headers to the more popular pin headers. The Metro M7 Airlift is the only Wi-Fi-enabled M7 board that we're aware of, so if you need both oodles of power and network connectivity, then this is a good choice. CircuitPython support is great, as you would expect of a board from Adafruit.

> **Having power to spare can make the build go a bit** smoother and lets you worry about optimisation later

This is the sort of board we like to use when prototyping projects. We might not need the raw performance or the dedicated networking hardware in the final build, but it's good to have it there while testing everything out and getting it all working. Yes, this is ridiculously powerful for a microcontroller, and yes, few of your projects really need this much grunt, but having power to spare can make the build go a bit smoother and lets you worry about optimisation later. ◻

VERDICT

A powerful board with Wi-Fi and great CircuitPython support.

10/10

PicoVision

Power your TV with Pico W

PIMORONI £34.50 | hsmag.cc/PicoVision

By Ben Everard

Connecting an HDMI display to a microcontroller is a pretty challenging task. The sheer rate of data that has to fly through the wires is daunting, and that's not taking into account any processing you might want to do on the data. RP2040 manages to do this with its programmable input/output system which attaches state machines to the I/O pins and can shuffle data out without any processor involvement. We're using the term 'HDMI' here because you plug in an HDMI cable and use it with an HDMI monitor. However, for the pedants among you, it is technically using DVI. HDMI is backwards-compatible with DVI, so this needn't concern you in use.

Right
You can combine text and images to create your frame. In this case, an info screen with data grabbed from the internet

RP2040 is limited by how fast it can run, so you'll probably find that PicoVision can't manage the full resolution of your display. PicoVision does go up to 1280×720, however, it does this by pushing both RP2040 and the HDMI protocol beyond their defined limits, so this won't work on all PicoVisions or all displays. 720×480 is a more reliable target. As well as a limited resolution, there's also a limited set of colours available. There's a set of images and GIFs on the PicoVision page which give a realistic idea of what's available (they match with our experience using the board).

PicoVision combines two microcontrollers – one RP2040 does the hard work of throwing data down the HDMI connection, leaving a second RP2040 (in the form of a Pico W) free to do whatever processing you want. These two interact using a pair of PSRAM buffers. The Pico W writes to one while the HDMI-RP2040 sends the other to the display.

Alongside the two microcontrollers and an HDMI port, there are three user buttons, a microSD card port, a STEMMA QT / Qwiic connector, and an I2S DAC for audio output. There is also the USB port which can (when being used with C++) be used in USB Host mode. You can attach a USB keyboard. In theory, most USB hardware can be connected, but in practice, it entirely depends on what information you can get about the hardware and how much effort you're willing to put into writing a driver for it.

The double RP2040 architecture means that you have one Pico W that you can use as you would any other Pico W. You have almost all the resources available for your program, with the exception of the

GPIO pins, most of which are in use. You can program this with either MicroPython or C++. In MicroPython, PicoVision is controlled by the display module in a similar way to other Pimoroni products – this lets you build up your output from shapes, sprites, and text. There's a good set of examples to show you how to use this. It's a bit rudimentary, but it's easy to use and works well.

This does all pose the question of what you would want an HDMI-enabled microcontroller for. Pimoroni obviously has something in mind as, alongside this board, they've announced a competition to create the best demoscene-style demo. In Pimoroni's words, this means: 'Demos are often characterised by their impressive graphical effects, compelling soundtracks, and the intricate interplay between visuals and audio, all typically constrained within specific hardware or file size limits. This constraint-driven creativity pushes sceners to extract every ounce of potential from a given platform'.

As well as giving geeks a chance to display their skills, PicoVision is a good option for anything that wants both a microcontroller and a medium-to-large display. You might mostly think of HDMI as being for monitors and TVs, but there's a wide range of HDMI displays from about 5 inches up, and they often come with mounting hardware for embedding in projects. We can't ignore that – at this price point – there are other things that can generate HDMI output and do a lot of other things at the same time, such as a Raspberry Pi Zero. PicoVision offers a very different set of things. Raspberry Pi Zeros offer a full Linux environment, which is either great or terrible depending on your use case. It's great if you need the power that this offers. However, if you don't need the power (you just want to put some images on a screen), it's a pretty terrible option because it comes with a huge amount of things that can go wrong, need updating, and generally require attention. By removing all that, PicoVision has the potential to be more understandable, more reliable, and also more robust.

> **PicoVision is a good option for anything that wants** both a microcontroller and a medium-to-large display

We suspect that a pretty large part of the market for PicoVision will be people who enjoy the simplicity of microcontroller programming and want to push it to its limits, for this is, by a significant margin, the easiest microcontroller board we've ever used for working with larger displays (beyond the SPI displays that go up to about 4 inches). If this is what you want to do, whether for practical reasons or just to see what performance you can squeeze out of the processor in the style of the demoscene, then this is by far the best option.

Above
There's a Pico W on the front and another RP2040 on the back

VERDICT

The best microcontroller board for controlling larger screens that we've used.

10/10

CROWDFUNDING NOW

Open UpCell

Make everything portable

$50 | hsmag.cc/OpenUpCell | Delivery: April 2024

Modern batteries are tiny, hold a lot of power, and are rechargeable. We've come to take these things for granted, but it really is miraculous. So much modern technology relies on the chemistry of lithium – from power tools to mobile phones, and even the laptop on which this review is being written. It's great to make use of this in our projects, however, so often we find the maker-friendly lithium battery hardware a bit limited for our needs. Many easy-to-use options are slow to charge and limited to small cells. There is good reason for an abundance of caution with lithium batteries – they're potentially a dangerous technology – mistreat the batteries, and there's a risk of a particularly noxious fire.

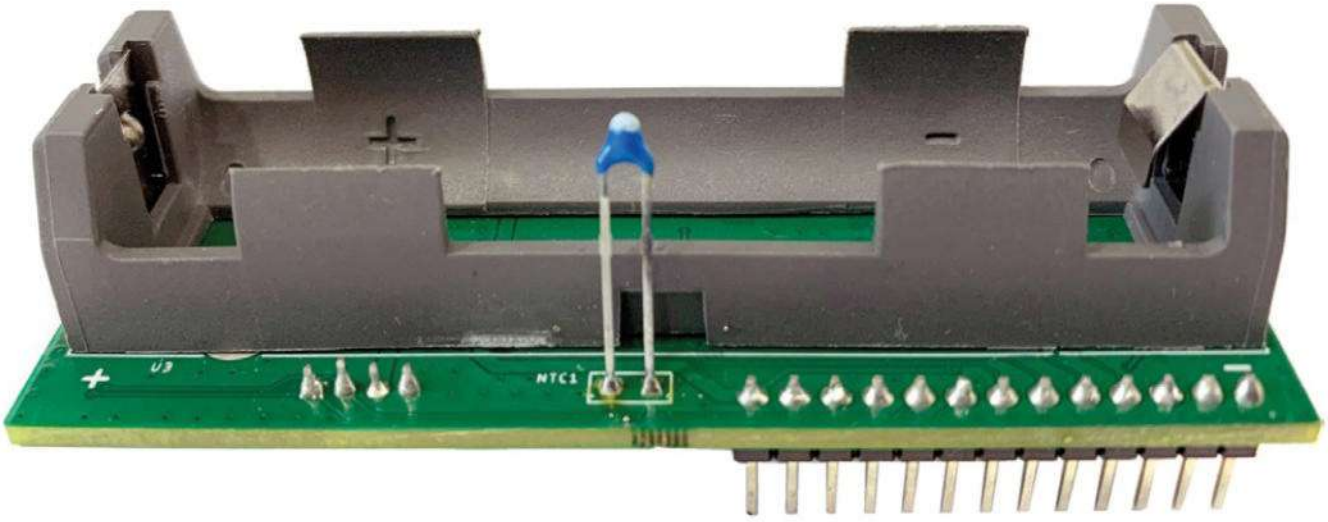
Open UpCell looks like it solves a lot of the problems – it enables fast charging with an off-the-shelf USB charger, it accepts a wide range of battery form factors, and it can output up to 3 A at 3 or 5 V.

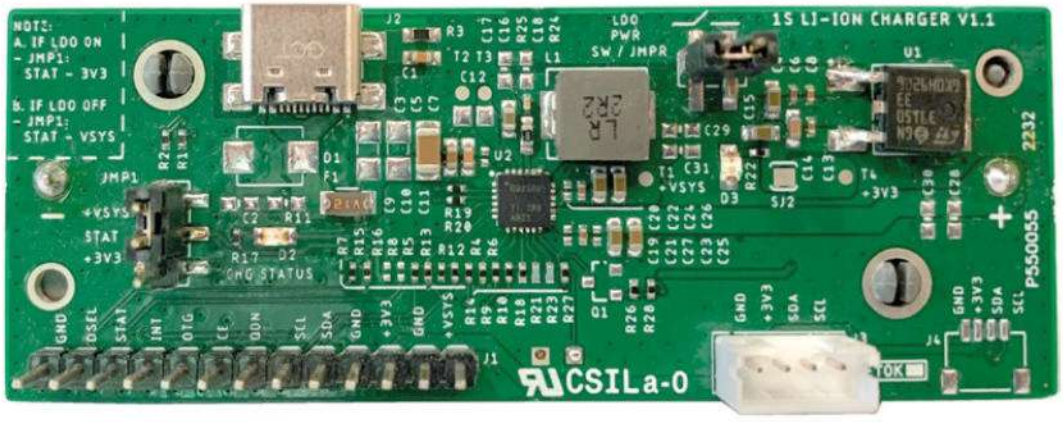
These are all great features and, as always, great features come at a cost, and in this case, it's $50. That's steep for a power supply, but if you need it, then it may save you a lot of pain and hassle.

We've not used it, so can't confirm if it actually works as promised but, potentially, this could make battery-powered projects a whole lot easier.

BUYER BEWARE

When backing a crowdfunding campaign, you are not purchasing a finished product, but supporting a project working on something new. There is a very real chance that the product will never ship and you'll lose your money. It's a great way to support projects you like and get some cheap hardware in the process, but if you use it purely as a chance to snag cheap stuff, you may find that you get burned.

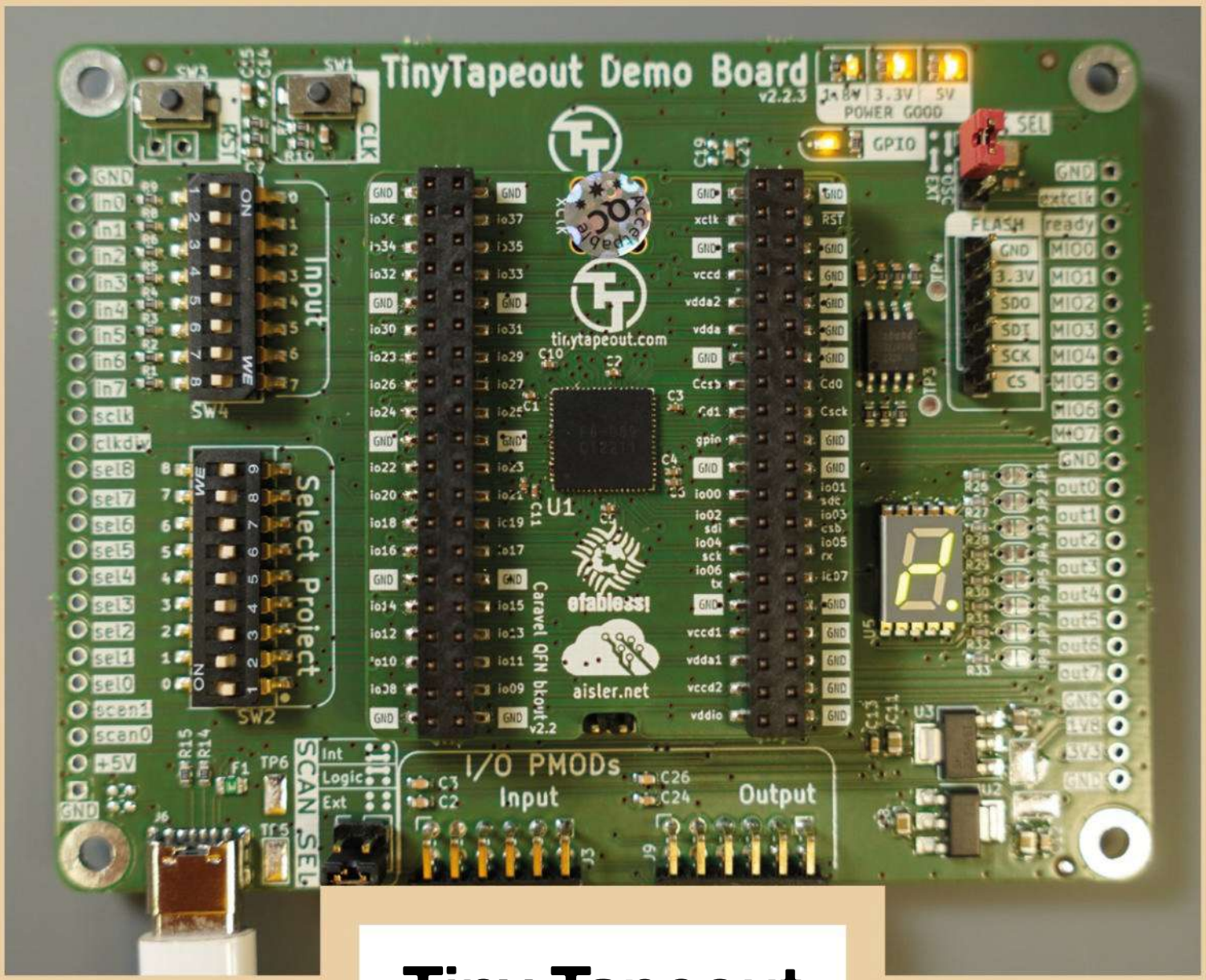




Above
Open UpCell includes a temperature sensor to help it charge safely

Left
The pinout includes an I2C connection for retrieving details about the battery's current state


next month
issue
#74
ON SALE
14 DECEMBER
DIY
GADGETS
ALSO
RASPBERRY PI
3D PRINTING
MUSIC
PCBS
AND MUCH MORE
DON'T MISS OUT
hsmag.cc/subscribe



Tiny Tapeout

The chip industry is incredibly complicated. Its raw materials are rare and hard to source, and must be refined several times over to get silicon pure enough to produce a usable wafer. At the highest end of the industry, the machines that engrave chip designs on these silicon wafers are made in only one factory, and cost up to $200,000,000 each. And again, at the cutting edge of chips, there's only one company that makes chips in significant numbers: TSMC – the Taiwan Semiconductor Manufacturing Company.

Despite this, you too can have a go at creating your own chip. At the centre of this board is a finished chip featuring designs by students on Matt Venn's Zero to ASIC course, using 100% open-source tools. We can't all have a chip manufacturing factory, but we can all be chip designers.

Your trust is our goal

From genuine, manufacturer-warrantied components to millions of in-stock parts shipped same day, be confident DigiKey will get you what you need—when you need it.

Visit digikey.co.uk today or call 0800 587 0991.

DigiKey

we get technical